Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

(повна назва факультету)

(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

(назва освітнього ступеня)

на тему: Розробка та захист веб додатку для формування онлайн системи питань та відповідей

Виконав(ла): студент(ка) 4 курсу, групи СБс-41
спеціальності 125 «Кібербезпека»

(шифр і назва спеціальності)

(підпис) Цимбалюк Г-О.Б. (прізвище та ініціали)

Керівник (підпис) Стадник М.А. (прізвище та ініціали)

Нормоконтроль (підпис) Лобур Т.Б. (прізвище та ініціали)

Завідувач кафедри (підпис) Загородна Н.В. (прізвище та ініціали)

Рецензент (підпис) Шимчук Г.В. (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)

Факультет _____
Кафедра _____

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) _____ (прізвище та ініціали)

« » _____

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня «Бакалавр»
(назва освітнього ступеня)

за спеціальністю 125 «Кібербезпека»
(шифр і назва спеціальності)

студенту Цимбалюку Глібу-Олександру Богдановичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка та захист веб додатку для формування онлайн системи питань та відповідей»

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від «___» _____ _____*року* № _____

2. Термін подання студентом завершеної роботи .

3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Аналітичний огляд існуючих рішень, технічне завдання, постановка задачі на розробку програмного забезпечення, опис та обґрунтування вибору структури та методу організації вхідних та вихідних даних, розробка алгоритму, визначення інформаційних зв'язків, написання текстів програм, аналіз нормативно-правової бази, розробка політик безпеки, технічний захист Інформації, вимоги ергономіки до організації робочого місця оператора ПК, дії оператора ПК під час повітряної тривоги в ситуації воєнного стану

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Пилипець М.І. д.т.н. професор кафедри МТ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ Цимбалюк Г-О.Б.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Стадник М.А.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дана кваліфікаційна робота присвячена розробці та захисту веб-додатку для створення онлайн системи питань та відповідей (Q&A). Метою роботи є проектування та реалізація цього веб-додатку з метою забезпечення зручного та ефективного обміну знаннями та інформацією.

У процесі дослідження був розглянутий базовий підхід, що використовується при розробці веб-додатків для систем Q&A. Аналізувалися приклади відомих систем Q&A з метою виявлення кращих практик та функціональних можливостей. Розробка веб-додатку здійснювалась у середовищі Rubymine з використанням мов програмування Ruby on Rails, JavaScript, розмітки Bootstrap та бази даних SQLite. Крім того, проведено детальний аналіз результатів роботи створеного веб-додатку для системи Q&A.

Отримані результати дослідження пропонується використовувати в якості методичного матеріалу розробниками програмного забезпечення під час проектування та розробки подібних веб-додатків. Розробка та захист веб-додатку для створення онлайн системи питань та відповідей має великий потенціал для покращення процесу навчання, співпраці та обміну знаннями в академічному, бізнесовому та загальному користувацькому середовищах.

Обсяг роботи: 84 сторінки А4 (без додатків), 22 посилання, 2 таблиці, 15 рисунків.

ЗМІСТ

ВСТУП

Розробка та захист веб-додатку для створення онлайн системи питань та відповідей (Q&A) є актуальною темою в сучасному інформаційному суспільстві. Завдяки стрімкому розвитку інтернету та електронних комунікацій, веб-додатки для обміну знаннями та спільного навчання набули великого значення для академічного середовища, бізнесу та широкого загалу користувачів.

Веб-додаток для створення онлайн системи питань та відповідей відкриває нові можливості для навчання, спілкування та обміну ідеями. Він дозволяє користувачам швидко знайти відповіді на свої запитання, отримати експертну думку та побачити різні точки зору на питання, що їх цікавлять. Такі системи стають основою для розвитку активного співтовариства, яке об'єднує людей з різних галузей знань та надає їм можливість ділитися своїми знаннями та досвідом.

Розробка веб-додатку для системи Q&A вимагає впровадження комплексного підходу, який включає аналіз потреб користувачів, проектування інтерфейсу, розробку бази даних та інтеграцію різноманітних функцій. Крім того, безпека є одним з найважливіших аспектів у розробці такого веб-додатку, оскільки він зберігатиме та оброблятиме чутливу інформацію користувачів. Тому, забезпечення конфіденційності, цілісності та доступності даних є невід'ємною частиною процесу розробки та захисту такого додатку.

Додаток для системи Q&A також має потенціал стати платформою для навчання та співпраці. Він може надавати можливість студентам, викладачам та дослідникам взаємодіяти, обмінюватися знаннями та залучатися до колективного навчання. Завдяки цьому, веб-додаток для системи Q&A сприяє академічному зростанню та створенню нових можливостей для наукових досліджень та інновацій.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Веб-додаток представляє собою програмний продукт, який використовується через веб-браузер. У цьому типі додатка браузер виступає в якості клієнта, а веб-сервер виконує роль сервера. Браузер може функціонувати як тонкий клієнт, зосереджуючи основну логіку додатка на сервері, а його завданням є відображення інформації, яка завантажується з сервера, та передача даних користувача. Браузер здатен відображати web-сторінки та зазвичай входить до складу операційної системи, хоча підтримка та оновлення його залежать від постачальника операційної системи. Однією з переваг такого підходу є незалежність клієнтів від конкретної операційної системи користувача, що робить веб-додатки межплатформеними сервісами. Ця універсальність та відносна простота розробки призвели до широкої популярності веб-додатків.

“AskIt” – це веб додаток який дасть змогу легко, швидко та ефективно створювати та відповідати на запитання. Програм з таким функціоналом в мережі є досить багато але нижче будуть приведені одні з самих популярних на 2023 рік:

“StackOverflow” - це веб-платформа для обміну знаннями, яка дозволяє користувачам задавати технічні питання, відповідати на них, а також редагувати та підтримувати якість вмісту. Сайт зосереджений на програмуванні та інших технічних питаннях, які пов'язані з розробкою програмного забезпечення.

StackOverflow працює на принципі вільного доступу, тобто всі питання та відповіді публікуються відкрито та доступні для перегляду користувачами з усього світу. Сайт забезпечує зручний пошук та сортування питань та відповідей, а також функції редагування та форматування тексту. StackOverflow заснований на принципі спільноти, де користувачі можуть переглядати відповіді та підписуватися на обговорення, щоб бути в курсі всіх новин.

Крім того, StackOverflow відрізняється високою якістю вмісту та активним товариством експертів з програмування та інших технічних питань. Сайт також забезпечує безпеку від несанкціонованого доступу та постійно оновлюється технічною командою, щоб забезпечити оптимальну роботу сайту та високу якість вмісту (див. рисунок 1.1).

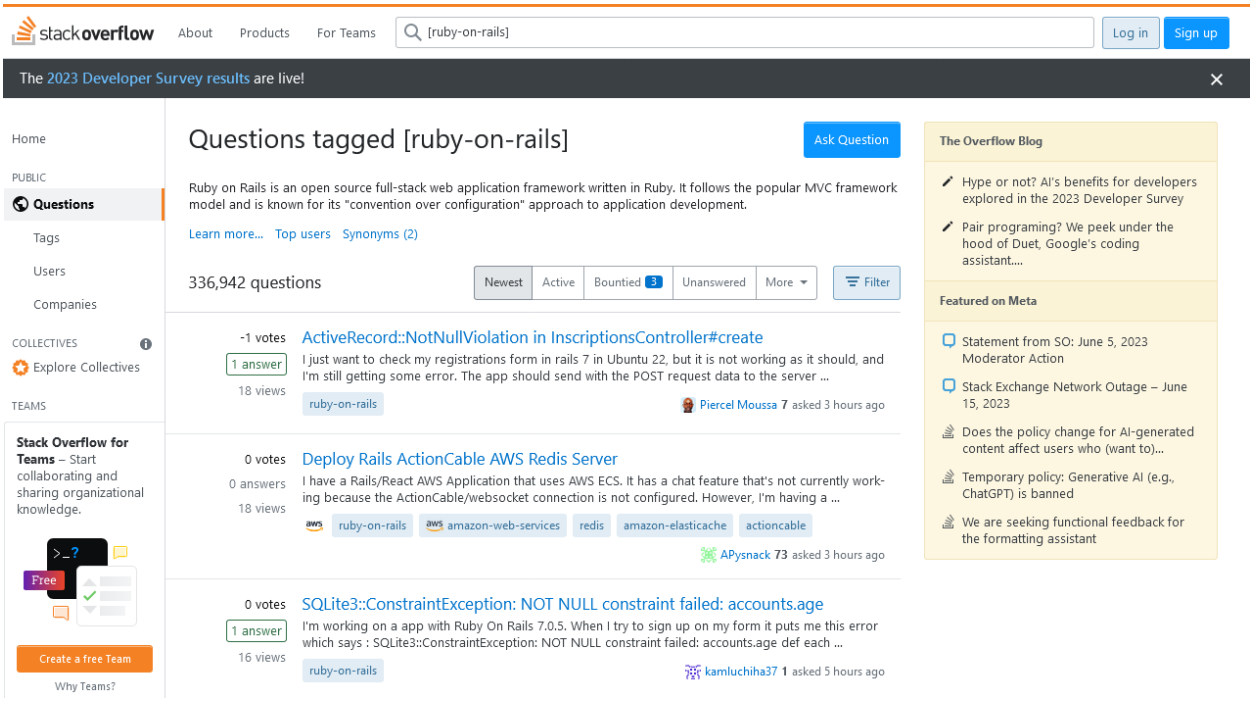

Рисунок 1.1 - Робоча область додатку “StackOverflow”

Основні функціональні можливості StackOverflow включають:

- Задавання питань: користувачі можуть задавати питання на будь-яку технічну тему, пов'язану з програмуванням та іншими технічними питаннями.
- Відповіді: користувачі можуть відповідати на питання, що були задані іншими користувачами.
- Редагування та покращення вмісту: користувачі можуть редагувати питання та відповіді, щоб зробити їх більш зрозумілими та корисними для інших користувачів.
- Рейтинг відповідей: користувачі можуть голосувати за відповіді, що найбільш точно відповідають на задане питання, тим самим визначаючи їхню якість.

- Пошук: користувачі можуть шукати питання та відповіді за ключовими словами та тегами.

- Теги: користувачі можуть додавати теги до питань та відповідей, щоб зробити їх більш зрозумілими та легкими для пошуку.

- Приватні повідомлення: користувачі можуть відправляти приватні повідомлення один одному, щоб отримати більш детальну відповідь на питання.

- Функції спільноти: користувачі можуть підписуватися на певні теги, стежити за питаннями та відповідями, а також приєднуватися до груп користувачів зі схожими інтересами.

- Режим нічного часу: користувачі можуть вибирати режим нічного часу для зручності читання та перегляду питань та відповідей в темному фоновому режимі.

“Reddit” - це соціальна мережа, де користувачі можуть ділитися контентом, створювати тематичні групи, обговорювати різні теми, голосувати за контент та коментарі.

Reddit має безліч підтематичних груп, які називаються "субреддитами". Кожен субреддит присвячений певній темі, і користувачі можуть додавати до нього відповідні пости та коментарі.

Основною функцією Reddit є голосування за контент, яке дозволяє користувачам оцінювати контент за його цікавістю та важливістю. Користувачі можуть голосувати за пости та коментарі, які їм сподобалися, а також проти них.

Reddit також має функцію "AMA" (Ask Me Anything), яка дозволяє користувачам ставити запитання експертам та знаменитостям. Крім того, на Reddit можна знайти новини, різні цікаві факти та матеріали на різні теми.

Reddit є платформою, що дозволяє користувачам ділитися контентом, обговорювати різні теми та голосувати за контент, що вони вважають важливим і цікавим (див. рисунок 1.2).

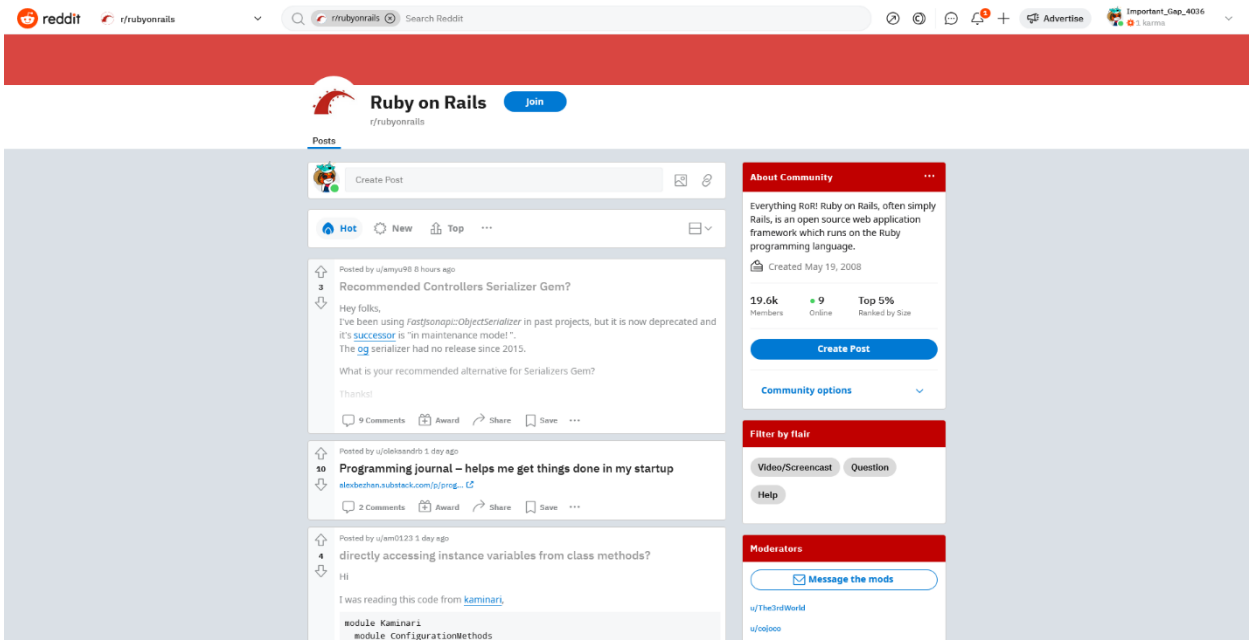

Рисунок 1.2 - Робоча область веб додатку “Reddit”

Основні функціональні можливості Reddit включають:

- Створення постів: користувачі можуть створювати пости на будь-яку тему, додавати до них тексти, зображення, відео та посилання на інші сайти.
- Створення коментарів: користувачі можуть створювати коментарі до постів та відповідати на коментарі інших користувачів.
- Підписка на субреддити: користувачі можуть підписуватися на тематичні субреддити, щоб отримувати повідомлення про нові пости та коментарі в цих групах.
- Голосування: користувачі можуть голосувати за пости та коментарі, використовуючи кнопки "вгору" та "вниз". Це дозволяє виділяти найбільш цікаві та важливі пости та коментарі.
- Функція "AMA" (Ask Me Anything): ця функція дозволяє користувачам ставити запитання експертам та знаменитостям, які добровільно відповідають на них у відповідних розділах.
- Пошук та фільтрація: користувачі можуть шукати пости за ключовими словами та фільтрувати їх за категоріями, часом публікації та іншими параметрами.

“Quora” - це соціальна мережа, де користувачі можуть задавати питання, шукати відповіді та обговорювати різноманітні теми, включаючи науку, технології, бізнес, культуру, політику та інше. Quora відрізняється від інших платформ соціального обміну інформацією тим, що основна увага зосереджена на контенті, який генерують користувачі.

Користувачі Quora можуть створювати профілі, додавати питання та відповіді, переглядати популярні теми, обговорювати новини та ділитися своїми знаннями з іншими користувачами. Більшість контенту на Quora генерується самими користувачами, що дає можливість отримувати різноманітні та високоякісні відповіді від експертів з різних галузей та областей знань.

Quora пропонує користувачам функції спільноти, щоб знайти та приєднатися до груп, які обговорюють конкретні теми, а також функцію підписки на відповіді та питання, щоб отримувати сповіщення про оновлення та нові відповіді. Крім того, Quora забезпечує можливість відкритого обговорення, де користувачі можуть взаємодіяти один з одним та ділитися своїми думками.

Quora створена для того, щоб допомогти користувачам дізнатися більше про різні теми та обговорювати їх з експертами та зацікавленими особами з усього світу (див. рисунок 1.3).

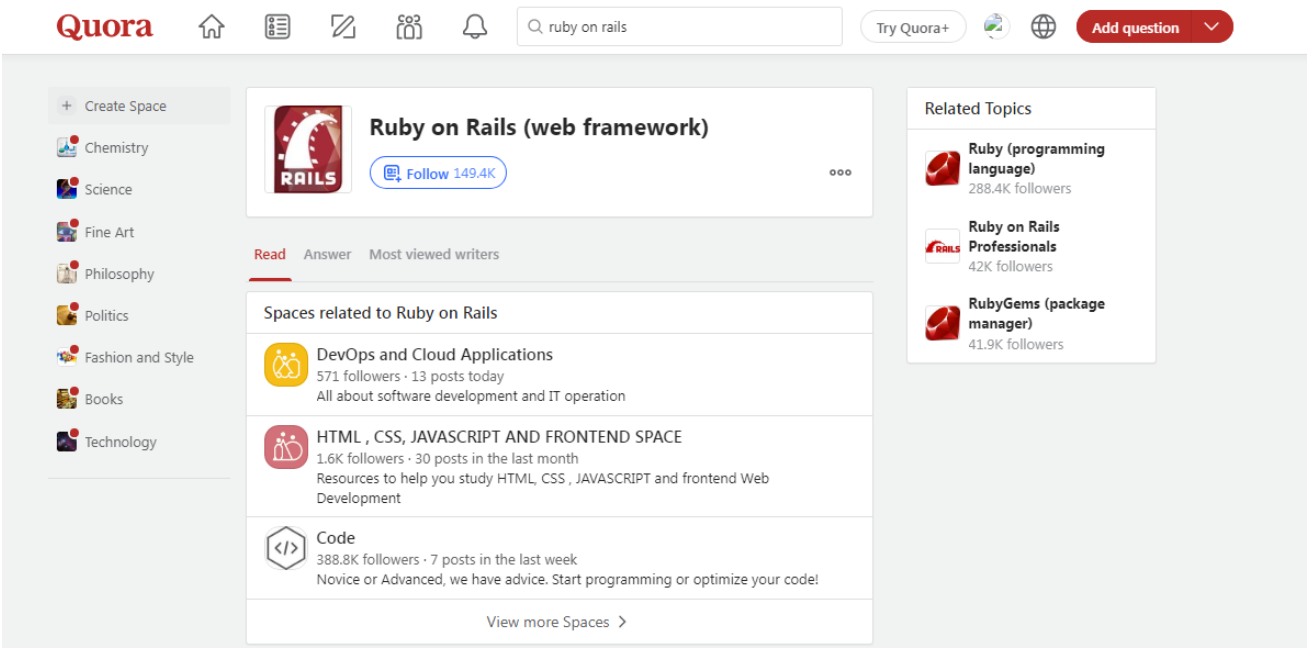

Рисунок 1.3 - Робоча область веб додатку “Quora”

Основні функціональні можливості Quora включають:

- Задавання питань: Користувачі можуть задавати різноманітні запитання на будь-яку тему.
- Відповіді: Користувачі можуть давати відповіді на запитання інших користувачів, а також взаємодіяти один з одним у коментарях.
- Пошук контенту: Користувачі можуть шукати контент, використовуючи ключові слова або теми.
- Обговорення: Користувачі можуть обговорювати різні теми у відкритих групах або спільнотах.
- Спільноти: Користувачі можуть створювати та приєднуватися до спільнот на певну тему, де можна обговорювати певну тему з експертами або зацікавленими особами.
- Відстеження питань та тем: Користувачі можуть відстежувати певні питання або теми, щоб отримувати сповіщення про нові відповіді або оновлення.
- Рейтинг відповідей: Користувачі можуть голосувати за якість відповідей та допомагати іншим користувачам знаходити найкращі відповіді.
- Профілі користувачів: Кожен користувач має свій власний профіль, де можна переглядати інформацію про користувача та його діяльність на Quora.

“Quandora” - це соціальна мережа, яка спрямована на покращення комунікації та обміну знаннями в організаціях. Quandora надає інструменти для створення внутрішніх соціальних мереж, де співробітники можуть створювати проекти, обговорювати проблеми, ділитися знаннями та досвідом, задавати питання та отримувати відповіді від колег (див. рисунок 1.4).

Quandora пропонує декілька корисних функцій, таких як відстеження запитів та відповідей, оцінка якості відповідей, розподіл прав на доступ до інформації та можливість створювати бази знань. Крім того, Quandora підтримує інтеграцію з іншими інструментами, такими як Google Drive, Dropbox, JIRA, Salesforce та інші.

Quandora допомагає покращити комунікацію та обмін знаннями в організації, збільшити ефективність роботи та підвищити якість продукту .

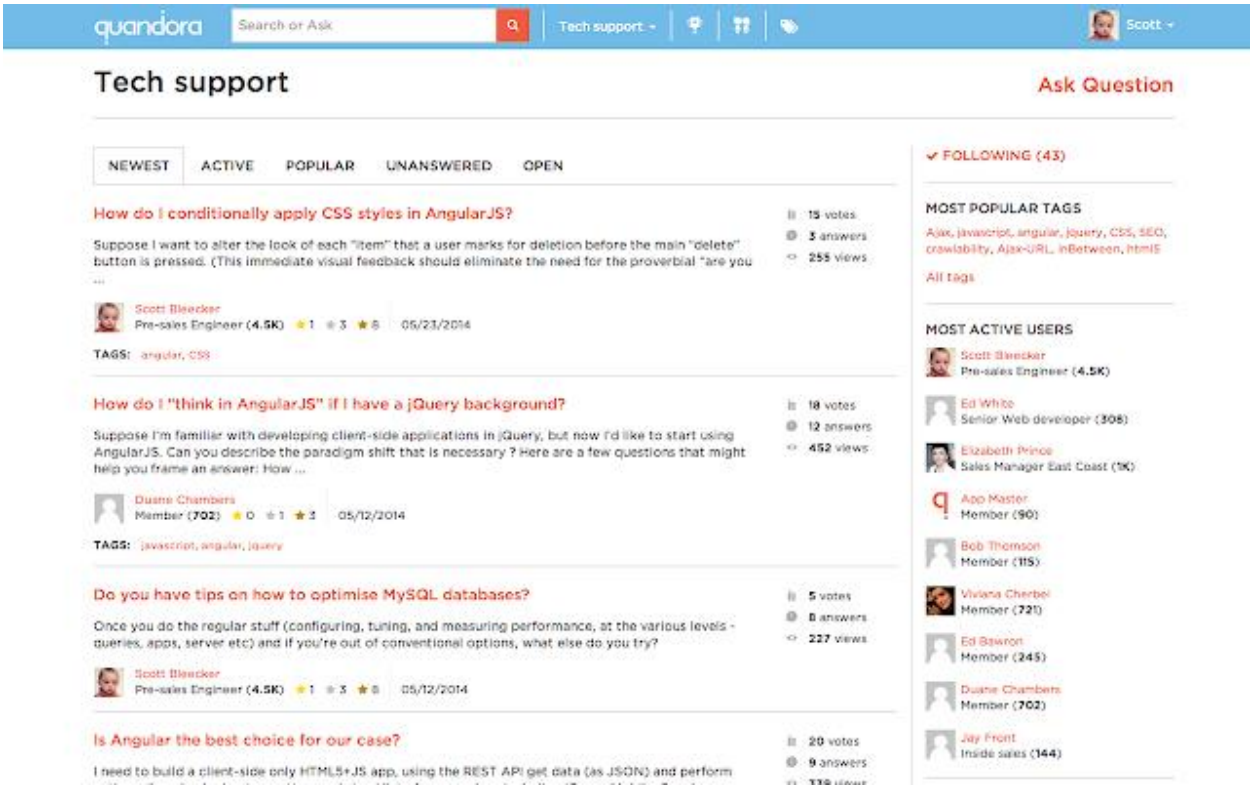

Рисунок 1.4 - Робоча область веб додатку “Quandora”

Основні функціональні можливості Quandora включають:

- Quandora - це соціальна платформа для обміну знаннями та спілкування між працівниками організації. Основні функціональні характеристики Quandora включають:

- Керування знаннями: Quandora дозволяє створювати, організовувати та керувати знаннями в організації. Користувачі можуть створювати нові статті, відповідати на запитання та використовувати функцію пошуку для знаходження потрібної інформації.

- Спільна робота: Quandora дозволяє користувачам працювати разом над проектами, обмінюватись ідеями та ділитись знаннями. Користувачі можуть коментувати та відповідати на питання інших користувачів.

- Аналітика: Quandora надає детальну статистику та аналітику щодо використання платформи, включаючи кількість запитань та відповідей, активність користувачів та рейтинги найактивніших користувачів.

- Інтеграція: Quandora може інтегруватись з іншими програмними засобами, такими як CRM, ERP та інші. Це дозволяє забезпечувати більш ефективний обмін знаннями та спілкування між працівниками організації.

- Безпека: Quandora забезпечує безпечний доступ до знань та інформації в організації. Кожен користувач має свій власний профіль, а доступ до конфіденційної інформації може бути обмежений за допомогою прав доступу та рівнів дозволу.

- Підтримка різноманітних форматів контенту: Quandora дозволяє додавати не тільки текстові повідомлення, але й графіки, фотографії, відео та аудіофайли.

- Підтримка спільнот: Quandora дозволяє створювати спільноти, де члени можуть обговорювати питання, ділитися досвідом та знаннями.

- Підтримка запитань та відповідей: Quandora дозволяє користувачам ставити запитання та отримувати на них відповіді від інших користувачів.

Ця функціональність дозволяє Quandora бути більш гнучкою та універсальною соціальною платформою для обміну знаннями та досвідом у порівнянні з іншими аналогами.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Темою кваліфікаційної роботи є розробка веб додатку для формування онлайн Веб-системи питань та відповідей (Q&A). Назва додатку – “AskIt”.

Веб-системи питань та відповідей (Q&A) є популярними інструментами, що надають онлайн-платформу для обміну знаннями та інформацією між користувачами. Ці системи створені з метою полегшити спілкування, спільну роботу та спільне розв'язання проблем шляхом постановки питань і отримання на них відповідей від інших учасників спільноти. Однією з веб-систем питань та відповідей, яка здобула велику популярність, є Stack Overflow.

Область застосування веб-системи питань та відповідей є доволі таки великою:

Програмування та розробка програмного забезпечення.

Веб-системи питань та відповідей широко застосовуються в галузі програмування та розробки програмного забезпечення. Розробники з усього світу використовують такі системи, наприклад, Stack Overflow, для обговорення проблем, отримання експертної допомоги, постановки запитань та обміну знаннями. Це дозволяє програмістам швидко отримувати відповіді на технічні питання, вирішувати проблеми та розвивати свої навички.

Комп'ютерна наука та дослідження.

Веб-системи питань та відповідей також знайшли своє застосування в галузі комп'ютерних наук та досліджень. Дослідники, вчені та студенти використовують ці системи для пошуку відповідей на актуальні запитання, обговорення проблем та розробки нових ідей. Веб-системи Q&A створюють сприятливу середу для колективного інтелекту та обміну ідеями, що сприяє розвитку та зростанню галузі комп'ютерних наук.

Технічна підтримка та консультування.

Однією з важливих областей застосування веб-систем питань та відповідей є технічна підтримка та консультування. Багато компаній та організацій використовують ці системи для надання підтримки своїм клієнтам та користувачам. Вони можуть створювати питання та отримувати швидкі та точні відповіді від фахівців у відповідних галузях. Це дозволяє підтримувати високий рівень задоволення клієнтів та забезпечувати ефективне вирішення проблем.

Навчання та освіта.

Веб-системи питань та відповідей також знайшли застосування в галузі навчання та освіти. Викладачі та студенти можуть використовувати ці системи для обговорення уроків, викликання запитань, обміну думками та спільного вирішення проблем. Це створює активну навчальну спільноту, в якій студенти можуть підтримувати один одного та отримувати додаткові пояснення та роз'яснення від викладачів або більш досвідчених студентів.

Веб-системи питань та відповідей відіграють значну роль у сфері програмування, комп'ютерних наук, технічної підтримки та навчання. Вони створюють місце для обміну знаннями, розвитку навичок та спільної роботи. Широкий спектр застосування цих систем робить їх важливими інструментами для співпраці та обміну інформацією у веб-середовищі. З ростом інформаційних технологій і залученням більшої кількості користувачів, веб-системи питань та відповідей мають потенціал для подальшого розвитку та покращення спільноти програмістів та фахівців в інших галузях.

1.2.2 Призначення розробки

Розробка веб-системи питань та відповідей є складним та багатогранним процесом, який має враховувати як експлуатаційне, так і функціональні вимоги до веб додатку.

Експлуатаційні вимоги визначають мету майбутнього використання веб додатку, тоді як функціональні вимоги вказують на методи досягнення цілі.

Експлуатаційне призначення.

Експлуатаційні призначення розробки веб-системи питань та відповідей полягає у створенні платформи, яка забезпечує зручну та ефективну взаємодію між користувачами, сприяє обміну знаннями та вирішенню проблем. Основні аспекти експлуатаційного призначення включають:

- Забезпечення спільноти: Веб-система питань та відповідей має створювати сприятливе середовище для об'єднання та взаємодії користувачів з різних галузей. Вона повинна привертати як нових користувачів, так і вже досвідчених фахівців, створюючи активну та динамічну спільноту.
- Полегшення обміну знаннями: Головна мета веб-системи питань та відповідей полягає у забезпеченні механізму для постановки питань та отримання на них відповідей. Вона повинна забезпечувати зручний спосіб пошуку, сортування та фільтрації існуючих запитань та відповідей.

- Розвиток та збереження знань: Розробка веб-системи питань та відповідей повинна сприяти збереженню та розвитку знань у різних галузях. Вона має створювати можливості для акумуляції знань, їх структурування та індексування, щоб користувачі могли легко отримати доступ до накопиченого досвіду та експертної інформації.

Функціональне призначення.

Функціональне призначення розробки веб-системи питань та відповідей визначає засоби досягнення поставленої мети, а саме створення зручної та ефективної платформи обміну знаннями. Основні функціональні вимоги до веб-системи питань та відповідей включають:

- Постановка питань: Веб-система повинна надавати можливість користувачам поставити питання, вказуючи відповідну галузь або тему. Це дозволяє залучати увагу фахівців, які мають відповідний досвід та знання.

- Відповіді та коментарі: Система повинна дозволяти користувачам відповідати на поставлені питання, давати коментарі та обговорювати рішення. Це стимулює активну участь спільноти та сприяє обміну різними поглядами та підходами.

- Оцінювання відповідей: Система повинна надавати можливість користувачам оцінювати якість та корисність отриманих відповідей. Це допомагає відокремити найкращі відповіді та надає важливу інформацію для майбутніх користувачів.

- Пошук та фільтрація: Веб-система повинна мати потужні механізми пошуку, сортування та фільтрації, щоб користувачі могли швидко знайти потрібну інформацію. Це включає можливість використовувати ключові слова, теги, категорії та інші параметри для точного його локації.

Розробка веб-системи питань та відповідей має як експлуатаційне, так і функціональне призначення. Експлуатаційне призначення включає створення спільноти, полегшення обміну знаннями та розвиток навчального середовища. Функціональне призначення включає постановку питань, надання відповідей, оцінювання, пошук та фільтрацію. Забезпечення цих функцій допомагає досягти

мети розробки веб-системи питань та відповідей - створення зручного та ефективного середовища для обміну знаннями та співпраці користувачів у різних галузях.

1.2.3 Вимоги до програмного забезпечення

Стандарт - загальні державні вимоги до програмного забезпечення затвердженні на законодавчому рівні. Існує велика різноманітність українських стандартів ДСТУ, міжнародних ISO/IEC/IEEE, RUP, SWEBOK та багато інших.

В них детально прописані рекомендації щодо етапів розробки, написання ТЗ, розробки ПЗ та подальшого його експлуатації.

Для даного кваліфікаційного проекту прописані наступні вимоги щодо:

1) Вхідних даних: додаток отримує інформацію введену користувачем.
2) Вихідних даних: отримані дані представляються у зручному для користувача вигляді.
3) Функціональні вимоги:

а) авторизація та аутентифікація користувачів: Система повинна забезпечувати механізми авторизації та автентифікації користувачів, щоб забезпечити безпеку та конфіденційність інформації. Користувачі повинні мати можливість створювати облікові записи, керувати ними та авторизуватись для доступу до функцій системи;

б) постановка питань: Система повинна дозволяти користувачам поставити питання, вказуючи відповідну галузь або тему. Користувачі повинні мати можливість додавати деталі та прикріплювати файли до своїх питань. Додавання тегів до запитань для полегшення пошуку;

в) відповіді та коментарі: Система повинна дозволяти користувачам відповідати на поставлені питання, давати коментарі та обговорювати рішення. Користувачі повинні мати можливість вказувати правильні відповіді та вибирати найкращі варіанти з пропонованих;

г) рейтинг та оцінювання: Система повинна надавати можливість користувачам оцінювати якість та корисність питань і відповідей. Користувачі повинні мати змогу ставити оцінки та залишати відгуки про якість матеріалів, що допоможе визначити найкращі ресурси та контент;

д) пошук та фільтрація: Система повинна мати потужні механізми пошуку, сортування та фільтрації, щоб користувачі могли швидко знайти потрібну інформацію. Пошук повинен підтримувати ключові слова, теги, категорії та інші параметри для точного локації релевантної інформації;

е) модерування та управління вмістом: Система повинна мати можливість модерувати та управляти вмістом, щоб забезпечити якість та відповідність питань та відповідей. Модератори повинні мати змогу видаляти неприпустимий або недостатньо якісний вміст та забезпечувати дотримання правил спільноти;

ж) багатомовність: Система повинна надавати можливість користувачам змінювати мову на якій буде відображатись сайт.

4) Вимоги до надійності:

а) ієрархічна структура комплексу: Програмне забезпечення повинно мати ієрархічну структуру, що дозволяє організувати комплекс системи в логічні блоки. Це сприяє зручному управлінню, масштабованості та модульності системи;

б) блоки контролю збійних ситуацій: В системі повинні бути вбудовані блоки, які контролюють та усувають можливі збійні ситуації. Наприклад, блок контролю наявності необхідних баз даних забезпечує, щоб система мала доступ до необхідної інформації для правильної роботи;

в) контроль вводу даних: Програмне забезпечення повинно мати механізми контролю вводу даних, що забезпечують стійкість системи до помилок користувача. Наприклад, контроль по діапазону значень даних допомагає уникнути неправильного вводу та забезпечити коректну обробку інформації;

г) збереження та відновлення даних: Програмне забезпечення повинно мати можливості зберігання та відновлення даних після аварійного переривання роботи. Це забезпечує безпечну та надійну роботу системи, зберігаючи важливу інформацію та уникнувши втрати даних;

д) обробка виняткових ситуацій: Система повинна мати механізми для обробки виняткових ситуацій, таких як не знайдено записів або некоректний запит. Це допомагає користувачам отримувати чіткі та зрозумілі повідомлення про помилки, що виникли, а також дозволяє системі відновлюватися до нормального стану після виникнення помилки;

е) захист від несанкціонованого доступу: Програмне забезпечення повинно мати механізми для захисту від несанкціонованого доступу до інформаційної бази. Це може включати введення пароля або інші методи автентифікації користувача, а також контроль доступу до функцій та даних системи;

ж) чіткі повідомлення про помилки: Програмне забезпечення повинно надавати чіткі та корисні повідомлення про місцезнаходження помилки та її характер. Це допомагає користувачам розуміти причини помилок і вирішувати їх, а також допомагає розробникам виявляти та виправляти проблеми у програмі.

1.2.4 Вимоги до програмної документації

Документація, що супроводжує програмне забезпечення, включає супроводжуючі документи, що висвітлюють загальні принципи, необхідні для передкористування. Цей вид документації забезпечує коректне використання наданого програмного забезпечення та пояснює його основні алгоритми.

Документація включає в себе:

- Специфікації, де вказується призначення та перелік файлів, включаючи документацію.

- Список власників оригіналів, який включає підприємства, що зберігають оригінали програмних документів. Цей перелік застосовується до великих та складних продуктів.

- Текст програми, де записутся коментарі з програмним кодом.

- Опис програмного продукту, що включає інформацію про його ієрархічну структуру та принципи функціонування.

- Програма і методика випробувань, де розміщений перелік і опис вимог, які повинні бути перевірені в ході випробування програми, методи контролю.

- Методика та програма тестування, яка містить перелік вимог, що підлягають перевірці під час випробування програми, а також методи перевірки їх виконання.

- Технічне завдання, яке представляє собою документ, що визначає призначення і сферу застосування програмного виробу, вимоги до його функціональності, етапи розробки та регламентує проведення тестувань.

- Пояснювальна записка, яка включає аргументацію вибраних технічних і техніко-економічних рішень, детальні схеми та описи алгоритмів, а також загальний огляд функціональності програмного виробу.

- Відомості про авторів та копірайти.

1.2.5 Техніко-економічні показники

Веб додаток розробляється на базі операційнї системи Linux з використанням робочого столу Ubuntu 22.04, мови програмування Ruby та фреймворком Ruby on Rails в середовищі Rubymine.

Тип інтерфейсу – графічний.

Мова інтерфейсу – українська, англійська.

Веб-сервер - це програмне забезпечення, яке надає доступ до веб-сторінок та інших ресурсів через Інтернет. Він відповідає за обробку запитів, надсилання відповідей та передачу даних між клієнтськими браузерами та веб-додатками.

Веб-сервер може обробляти різні типи запитів, такі як HTTP-запити, які використовуються для доступу до веб-сторінок, а також інші протоколи, які дозволяють взаємодіяти зі спеціалізованими додатками та послугами.

Для розробки програмного забезпечення сумарно повинно бути потраченим не більше 200 машино-годин та 10 людино-годин.

Програмне забезпечення поставляється у вигляді встановленого програмного забезпечення на веб сервер з відкритим кодом, з посиланням на репазотирій з повним кодом на таких платформах як GitHub та GitLab .

1.2.6 Стадії та етапи розробки

Відповідно до Державних стандартів України розрізняються такі стадії розробки програмного забезпечення:

- Формування вимог до системи (замовник визначає основні вимоги для програмного забезпечення).
- Розробка концепції системи (проводяться науково-дослідницькі роботи по сфері застосування та визначаються підходи до розробки).
- Розробка технічного завдання (створюються основні вимоги до системи).
- Розробка ескізного проекту (розробляється структура вхідних та вихідних даних та загальний алгоритм).
- Розробка технічного проекту (розробляється система).
- Написання програмного забезпечення.
- Тестування програмного забезпечення
- Реінжиніринг програмного забезпечення.
- Фінальне тестування.
- Встановлення на сервер.

- Розробка програмної документації.
- Введення в експлуатацію.

1.2.7 Порядок контролю та прийому

Для тестування працездатності програми було обрано два методи тестування: ручне (QA) та автоматизоване - Unit та Integration тести.

Для написання Unit тестів використовується модуль RSpec, який базується на перевірці коректної роботи всіх функціональних можливостей програмного забезпечення. Ці тести спрямовані на перевірку правильності внутрішньої логіки та взаємодії компонентів програми. Вони перевіряють прийом вхідних даних, обробку інформації та відповідність очікуваним результатам, не звертаючи уваги на внутрішню реалізацію.

Integration тес

ти використовують модуль Capybara, який дозволяє симулювати дії користувача та перевіряти взаємодію програми з реальним середовищем. Ці тести перевіряють правильну роботу програми в цілому, включаючи взаємодію між різними компонентами та перехід між сторінками. Вони можуть виявляти проблеми, які не були помічені під час окремого тестування компонентів.

Завершення розробки даного додатку означає, що усі тести успішно пройдені, функціонал працює належним чином та задовольняє вимоги технічного завдання. Однак, якщо під час реалізації функцій програми виникають ускладнення або змінюються вимоги, може знадобитися коригування технічного завдання для врахування нових вимог або змін.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЕКТУ

2.1 Постановка задачі на розробку програмного забезпечення

Основним завданням кваліфікаційної роботи є “Розробка та захист веб додатку для формування онлайн системи питань та відповідей” під назвою “AskIt” який буде виконувати наступні функції:

1) Створення запитання:
    а) запитання повинно мати заголовок і вміст;
    б) під час створення запитання система повинна зберегти інформацію про автора та дату створення.
2) Оновлення запитання:
    а) автор запитання може змінити заголовок і вміст запитання.
3) Видалення запитання:
    а) автор запитання може видалити своє запитання разом з усіма пов'язаними з ним відповідями.
4) Додавання відповіді до запитання:
    а) користувач може додати відповідь до конкретного запитання.
5) Оновлення відповіді:
    а) автор відповіді може змінити її зміст.
6) Видалення відповіді:
    а) автор відповіді може видалити свою відповідь.
7) Позначення запитання як вирішеного:
    а) автор запитання або модератор можуть позначити запитання як вирішене.
8) Реєстрація та автентифікація користувачів:
    а) користувачі можуть створювати облікові записи на сайті;
    б) користувачі можуть увійти до свого облікового запису.

9) Відновлення паролю:
    а) користувачі можуть використовувати функцію відновлення паролю для зміни забутого пароля.
10) Пошук та фільтрація запитань:
    а) користувачі можуть шукати запитання за заголовком або вмістом;
    б) користувачі можуть застосовувати фільтри до запитань (наприклад, за датою створення або рейтингом).
11) Отримання сповіщень про нові відповіді на запитання:
    а) користувачі, які підписалися на конкретне запитання, отримують сповіщення про нові відповіді.
12) Отримання сповіщень на електронну пошту:
    а) користувачі можуть налаштувати отримання сповіщень про нові відповіді на електронну пошту.

Програма реалізована на мові програмування Ruby та фреймфорку Ruby on rails у середовищі програмування Rubymine.

2.2 Опис та обґрунтування вибору структури та методу організації вхідних та вихідних даних

Даний кваліфікаційний проект розробляється на базі MVC фреймворку Ruby on Rails з використанням об'єктно-реляційної бази даних PostgreSQL.

MVC – Model-View-Controller це метод організації даних який передбачає поділ системи на три взаємопов'язані частини: View (інтерфейс користувача), Controller (модуль керування) та Model (модель даних)(див.рис.2.1).

Контролер відповідає за управління запитами, що надходять від користувача у форматі HTTP, таких як POST та GET (DELETE, PUT, PUTCH, DELETE), коли користувач взаємодіє з елементами інтерфейсу з метою виконання різних дій. Його головною функцією є виклик та координація

необхідних ресурсів та об'єктів, необхідних для виконання вказаних користувачем дій. Контролер взаємодіє з відповідною моделлю для виконання потрібних завдань і вибирає відповідне представлення.

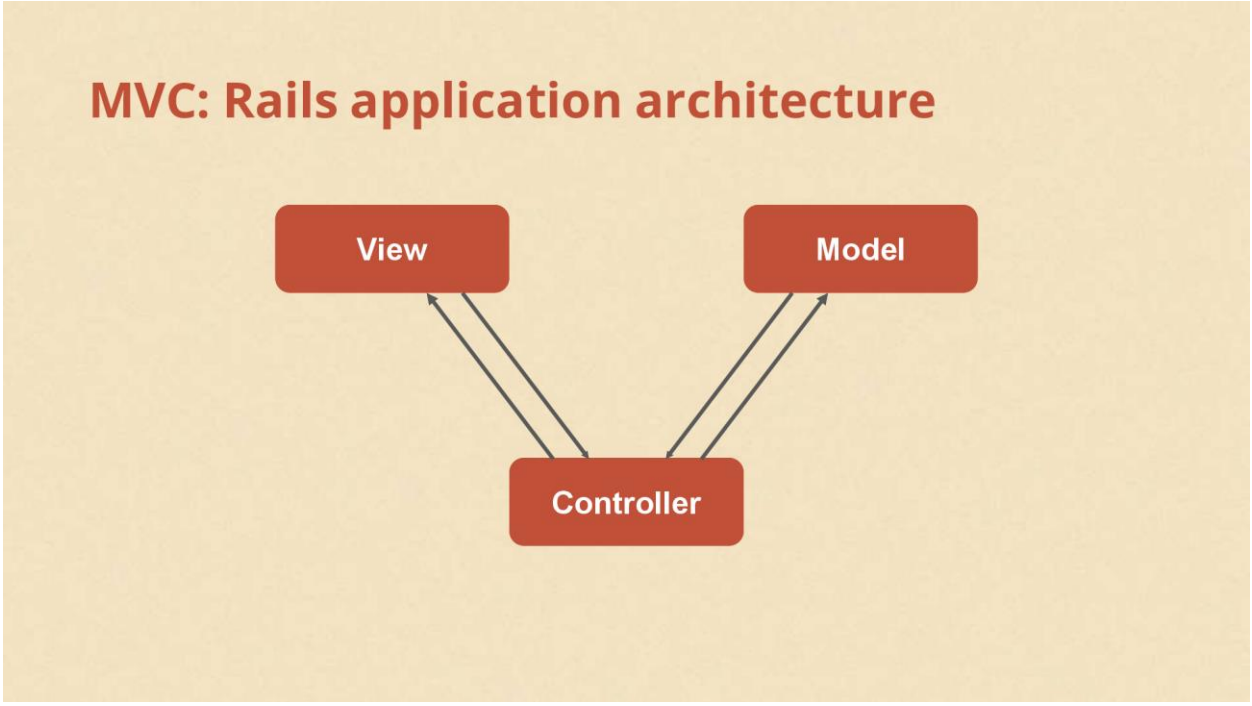

Рисунок 2.1 - Загальна концепція MVC

Модель управління додатком є комплексною системою, яка включає в себе дані та набір правил, необхідних для ефективної обробки цих даних. У кожній програмі структура моделі визначається як набір даних, що підлягають певному обробленню. Що означає "користувач" для даного додатку? Це просто набір даних, які мають бути оброблені згідно з встановленими правилами, наприклад, дати не можуть вказувати на майбутнє, електронна адреса повинна мати відповідний формат, а ім'я не може перевищувати задану кількість символів і так далі.

Модель надає контролеру концепцію про запитані користувача, які включають повідомлення, сторінки та інші форми. Модель даних залишається незмінною, незалежно від способу їх подання користувачеві. Таким чином, ми маємо можливість вибрати будь-який доступний спосіб відображення цих даних.

Модель, в рамках додатку, включає найзначущу логічну складову, що вирішує поставлені завдання. Вона зосереджена на розробці та реалізації логічних алгоритмів, необхідних для досягнення поставлених цілей. З іншого боку, контролер має на увазі переважно організаційну логіку, що відповідає за управління додатком у цілому. Його функції охоплюють координацію роботи різних компонентів, забезпечення потрібних ресурсів та взаємодію з моделлю для досягнення ефективного функціонування програмного продукту.

Потрібно уточнити, що в даному випадку описаний підхід з «товстої» моделлю і «тонким» контролером. Дуже часто практикується підхід навпаки - «Тонка» модель і «товстий» контролер - коли бізнес-логіка укладена в контролері, а модель є лише даними.

Модель робить так званий мапінг - перенесення однієї поведінки на іншу, в даному випадку можливо писати код на ruby замість sql. Також дозволяє використовувати будь-яку базу даних з тих що підтримує ActiveRecord.

ActiveRecord є фреймворком ORM, що представляє собою методику об'єктно-реляційного відображення (ORM), що використовується для зв'язку складних об'єктів додатків з таблицями в системах управління базами даних.

ORM надає зручний спосіб збереження та отримання властивостей та взаємозв'язків об'єктів додатків у базі даних, уникаючи необхідності написання прямих SQL-виразів. Це дозволяє зменшити загальний обсяг коду, необхідного для доступу до бази даних. В конфігураційному файлі ми вказуємо тип бази даних, з якою працюємо, і ActiveRecord вміє автоматично перетворювати наш код в відповідний код бази даних, що використовується (config/database.yml).

Представлення відповідає за різноманітні способи відображення даних, які отримані з моделі. Воно може мати форму шаблону, що заповнюється отриманими даними. Також можуть існувати різні види представлень, і контролер обирає той, який найкраще відповідає поточній ситуації.

Веб-додаток, у своїй загальній структурі, зазвичай складається з набору контролерів, моделей та представлень. Контролери можуть бути організовані у

вигляді основного компонента, що приймає всі запити і, залежно від контексту, викликає інші контролери для здійснення відповідних дій.

Однією з основних переваг, яку ми отримуємо застосовуючи концепцію MVC, є чітке розділення логіки інтерфейсу користувача від логіки програми. Цей поділ дозволяє нам легше управляти та підтримувати систему, розподіляючи відповідальності між компонентами та забезпечуючи більшу модульність і перевикористовування коду.

Підтримка різних типів користувачів, які використовують різні пристрої, є важливою проблемою в сучасних умовах. Розробка інтерфейсу повинна враховувати відмінності між персональними комп'ютерами та мобільними телефонами. Модель забезпечує однакові дані для обох випадків, але вибір відповідного представлення даних залежить від контролера.

Концепція Модель-Вид-Контролер (MVC) суттєво спрощує розробку складних додатків. Код стає більш структурованим, що полегшує підтримку, тестування та повторне використання рішень (див. рис. 2.2). Такий підхід забезпечує збалансований та організований розподіл функцій між компонентами системи, що сприяє покращенню якості програмного продукту та забезпеченню зручності його використання.

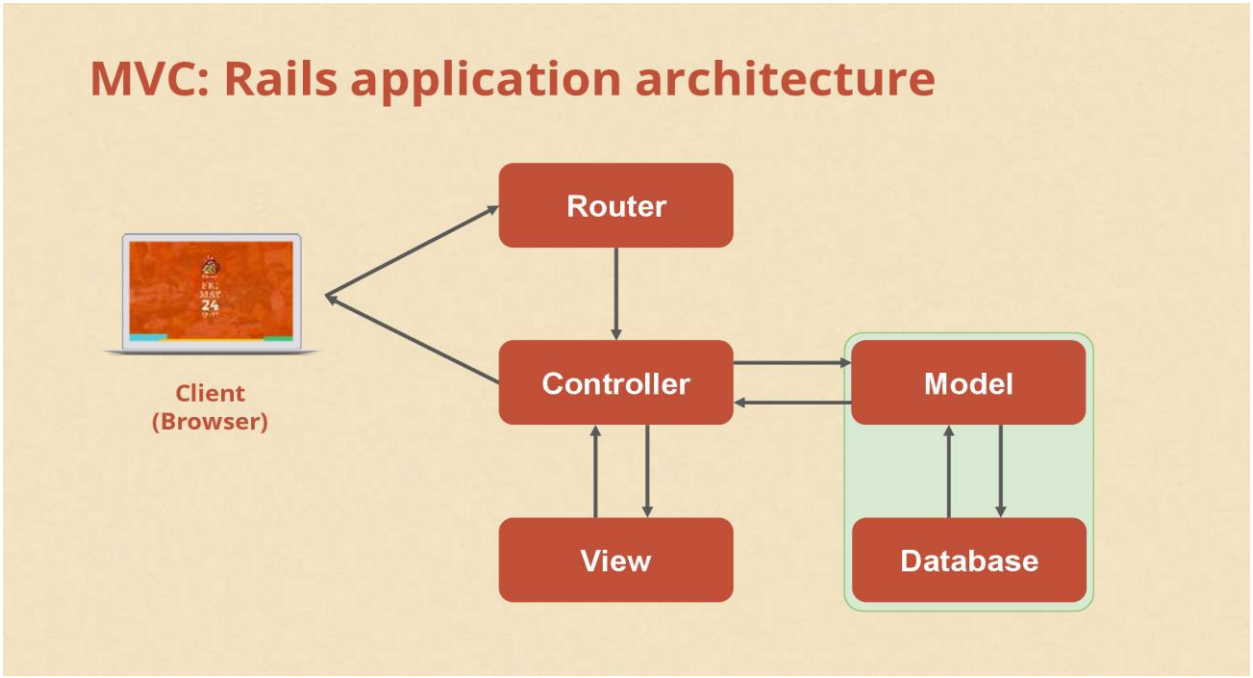

Рисунок 2.2 - Обробка запиту в MVC

2.3 Розробка алгоритму

2.3.1 Зовнішнє проектування програми

Розроблена система матиме веб інтерфейс, який реалізується засобами MVC - View.

View надає можливість підключити зовнішні бібліотеки та фреймворки для формування зовнішнього вигляду сторінки. Для даної кваліфікаційної роботи був використаний фреймворк Bootsrap версії 5.1. Bootstrap - це потужний фреймворк для розробки веб-додатків, який містить набір готових компонентів, стилів і скриптів. Він дозволяє розробникам дволі-таки швидко створювати веб сторін адаптивних веб додатків в Інтернеті.

View складається з 3 частин:

Layouts (пер. макет) – Це точка входу, контейнер для шаблонів. Формує головну структуру шаблону сторінки, тут підключається розмітка яка постійно повторюється і та яка має бути на всіх сторінках в проекті, наприклад: navbar, header, footer.

Template (пер. шаблон) – Це представлення екшенів контроллером. Відповідає за основний контент на сторінці. В шаблоні розробляється розмітка та підключаються партіали.

Partials (пер. частина) – Це розбиття великої кількості коду на маленькі частини, це дає можливість перевикористання партіалів з різним змістом (інформацією) але з одинаковою розміткою. Контент партіалів може бути відрендереним в любій частині шаблону.

Головна структура шаблону находится в файлі application.html.erb.
(див. Додаток А). В даний файл підключаются javascript, стилі, шрифти та встановлюється розмітка.

Реалізація на мові програмування.

Лістинг 2.1

```
<%=  stylesheet_link_tag  'application',  media:  'all',  'data-
turbolinks-track': 'reload' %>
      <%=  javascript_pack_tag  'application',  'data-turbolinks-
track': 'reload' %>
       <link
href="https://fonts.googleapis.com/css2?family=Roboto+Mono:wght@300
;
      400;700&display=swap" rel="stylesheet">
```

Структура шаблону головної сторінки додатку матиме вигляд, представлений на рисунку 2.3

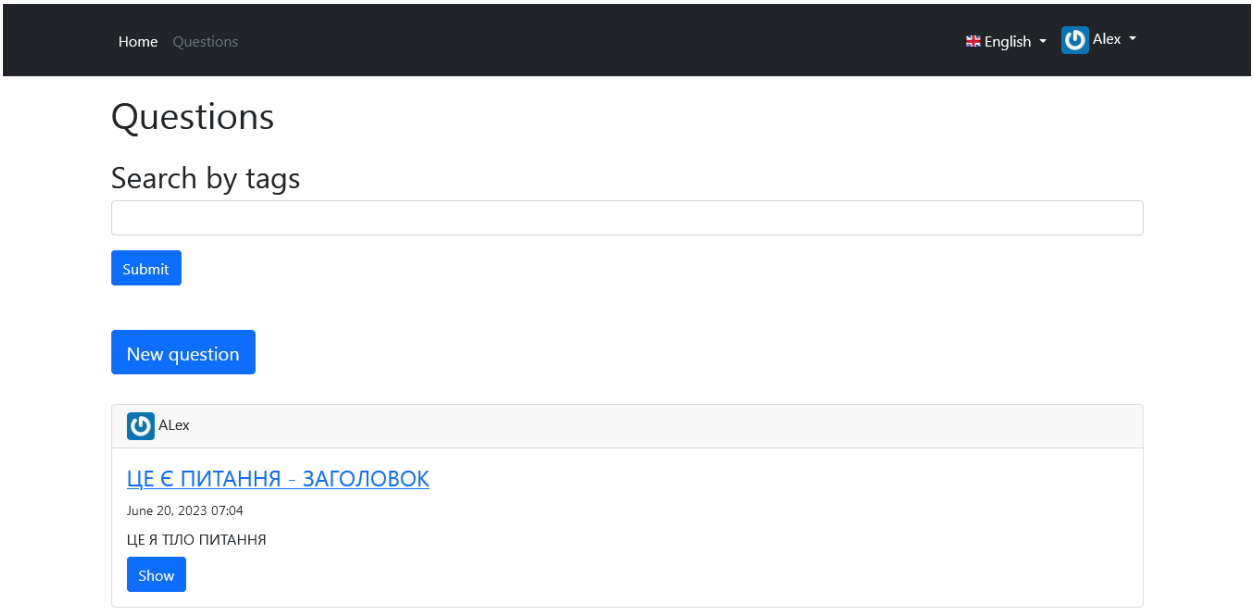

Рисунок 2.3 - Структура шаблону головної сторінки

Формування сторінки проходить в 4 основних етапа.

Загрузка меню Navigation (див. рис. 2.4) що находиться в файлі application.html.erb. Якщо користувач не залогінився, тоді меню містить кнопки для переходу на інші сторінки, випадаючий список для зміни мови, кнопку реєстрації та кнопку входу.

Реалізація на мові програмування.

Лістинг 2.2

```
      <nav class="navbar navbar-expand-lg navbar-light bg-white
border-bottom shadow-sm">
      <%= link_to icon('fas', 'list-ul', 'TodoList'), root_path,
class: 'navbar-brand' %>

      <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarNavAltMarkup" aria-
controls="navbarNavAltMarkup" aria-expanded="false" aria-
label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse"
id="navbarNavAltMarkup">
      <div class="navbar-nav ml-auto">
      <%= link_to icon('fas', 'home', 'Home'), todo_lists_path,
class: 'nav-item nav-link' %>
      <%= link_to icon('fas', 'plus', 'Add List'),
new_todo_list_path, class: 'nav-item nav-link mr-3' %>
      <%= link_to icon('fab', 'github', 'GitHub'),
'https://github.com/Leeroy6821/TodoList', target: "_blank", class:
'nav-item nav-link' %>
      <%= link_to icon('fab', 'gitlab', 'GitLab'),
'https://gitlab.com/Leeroy6821/todolist', target: "_blank", class:
'nav-item nav-link' %>
      </div>

      <% if user_signed_in? %>
      <%= link_to 'Sign out', destroy_user_session_path, method:
:delete, class: 'btn btn-dark ml-md-4' %>
      <% else %>
      <%= link_to 'Log in', new_user_session_path, class: 'btn btn-
dark ml-md-4' %>
      <%= link_to 'Sign up', new_user_registration_path, class:
'btn btn-dark ml-md-2' %>
      <% end %>
      </div>
      </nav>
```

Рисунок 2.4 - Меню навігація неавторизованого користувача

Якщо користувач ввійшов у систему, тоді кнопки в навігаційному меню змінюються. Замість кнопки вхід та вихід з системи, появляється фото користувача та випадаючий список де користувач може переглянути свій профіль або вийти з системи (див.рис 2.5)

Рисунок 2.5 - Меню навігація авторизованого користувача

Вслід за навігацією загружається Основна частина сторінки. Для неавторизованих користувачів сторінка включає в себе назву розділу “Question”, поле для пошуку та запитання користувачів оформлених в виді карточок (див. рис. 2.6).

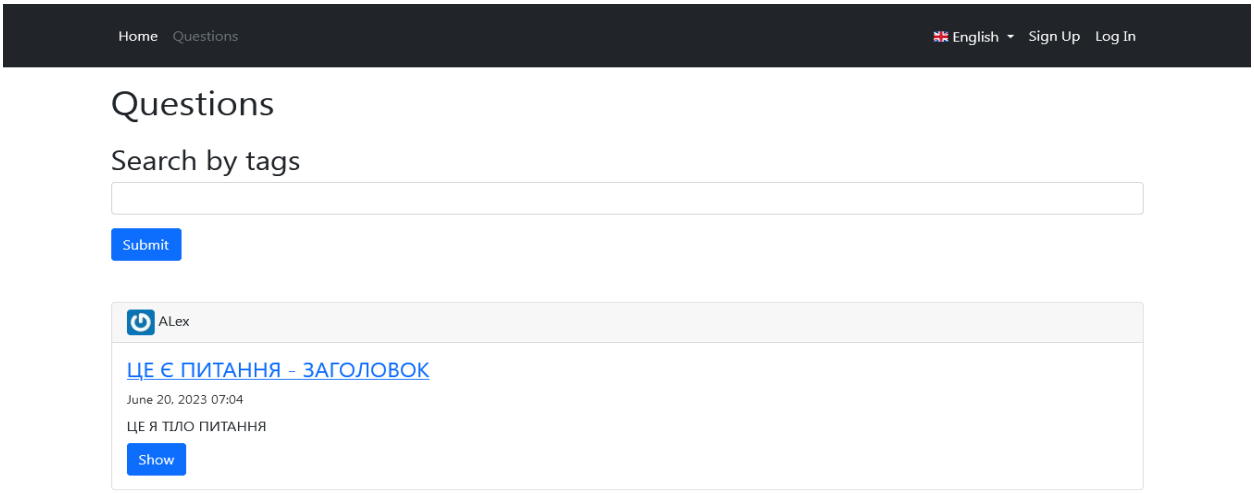

Рисунок 2.6 – Основна частина сторінки неавторизованого користувача

Для авторизованих користувачів добавляється кнопка для створення нових запитань. (див.рис 2.7).

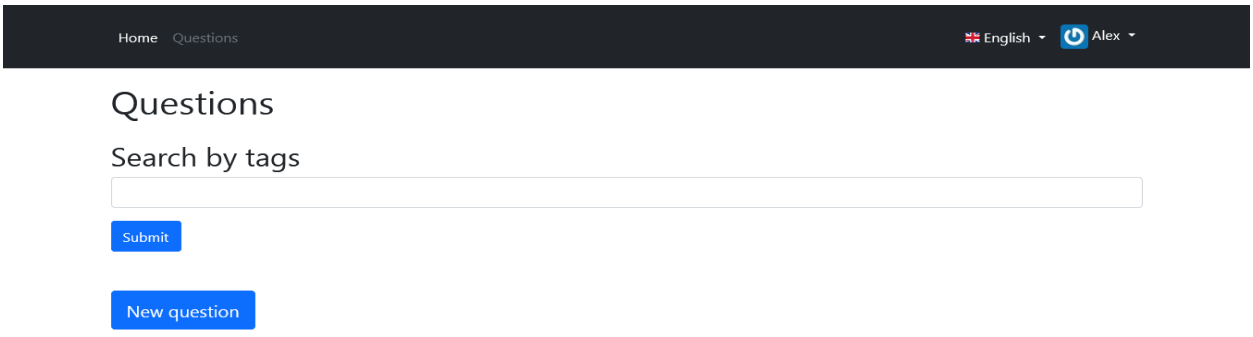

Рисунок 2.7 – Основна частина сторінки авторизованого користувача

Запитання користувачів оформленні в виді карток. Картка містить заголовок запитання, дату створення, тіло запитання (не більше 100 символів) та кнопки “show”, “edit”, “delete” які дають змогу користувачеві дивитись, редагувати та видаляти запитання (див. рис 2.8).

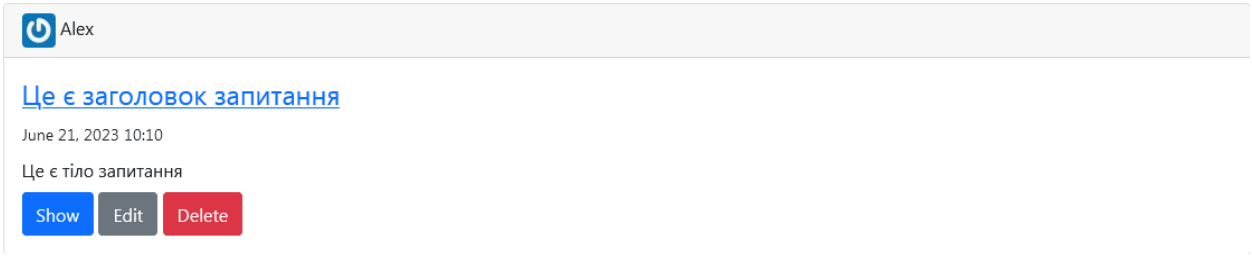

Рисунок 2.8 – Структура запитання

Реалізація на мові програмування.

Лістинг 2.3

```
<article class="card my-3">
  <section class="card-header">
    <%= question.user.gravatar %>
    <%= question.user.name_or_email %>  </section>

   <div class="card-body">
    <h4><%= link_to question.title, question_path(question) %></h2>

 <section class="card-text">
       <%= tag.time datetime: question.formatted_created_at do %>
         <small><%= question.formatted_created_at %></small>
       <% end %>

       <div class="my-2">
         <%= render question.tags %>
       </div>
 <p class="my-2">
```

Продовження лістингу 2.3

```
 <%= truncate strip_tags(question.body), length: 150, omission:
t('global.text.omission') %>
      </p>
    </section>
 <%= link_to t('global.button.show'), question_path(question),
class: 'btn btn-primary' %>

    <% if policy(question).edit? %>
      <%= link_to t('global.button.edit'),
edit_question_path(question), class: 'btn btn-secondary' %>
    <% end %>
  </div>
</article>
```

Сторінка запитання містить в собі заголовок запитання, дату створення запитання, повне тіло запитання, профіль користувача що його створив, кнопки редагування та видалення запитання, поле для вводу коментарів та відповідей (див. рис. 2.9)

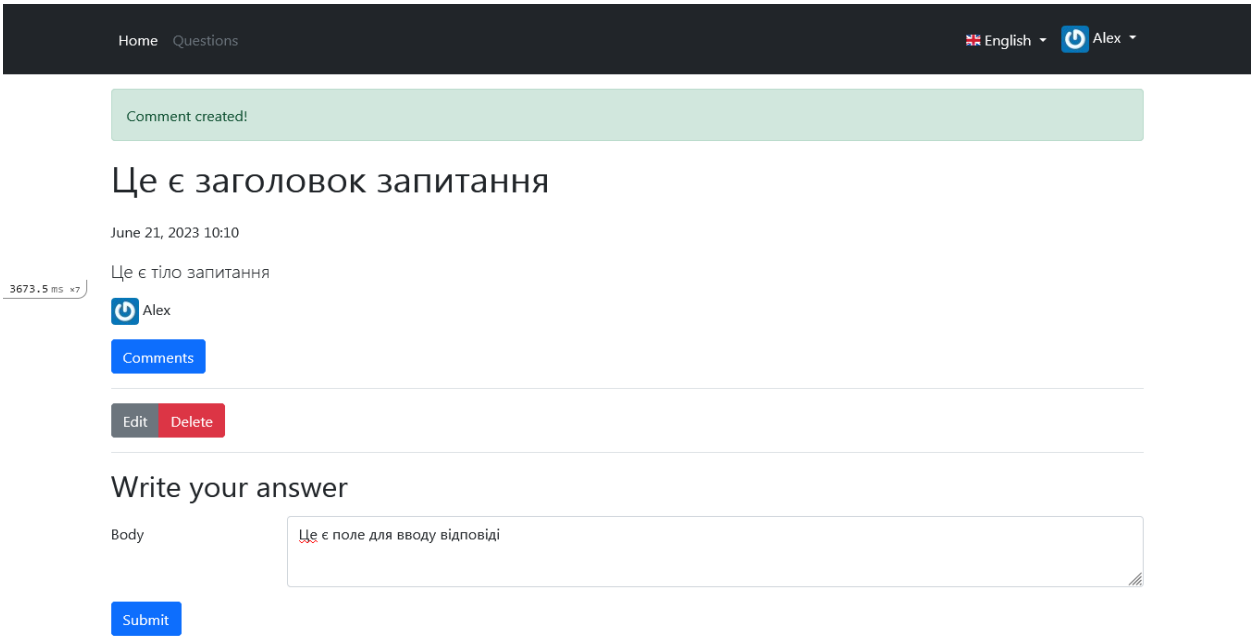

Рисунок 2.9 – Структура сторінки з запитанням

2.3.2 Проектування логіки програми

Розглянемо схему роботи додатку, представлену на рисунку 2.10

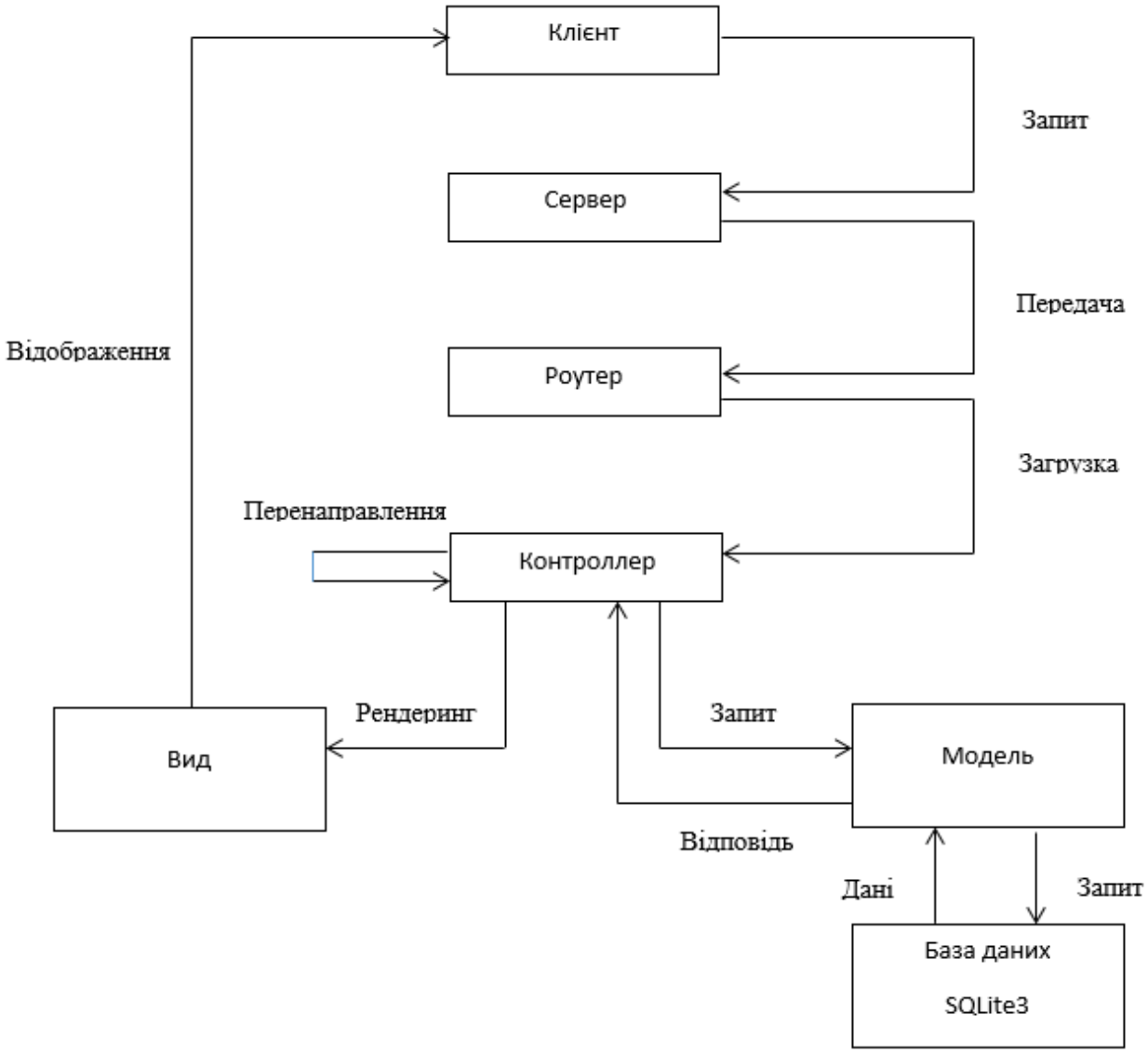

Рисунок 2.10 – Схема роботи додатку

Клієнт відправляє HTTP запит на сервер, сервер обробляє і зберігає ці дані. Дальше роутер розпізнає посилання які приходять з HTTP запиту і відправляє дальше на екшени і контроллери. Контроллер посилає запит в модель, модель посилає запит в базу даних , база даних відправляє потрібну інформацію моделі, модель повертає дані контроллеру. Після цього контроллер посилає запит в View, View рендерить і відображає користувачу.

2.4 Визначення інформаційних зв’язків

Оскільки для роботи системи важливе значення мають інформаційні звязки різних частин та модулів, розглянемо їх детальніше. При цьому слід враховувати що використовується парадигма MVC, і в нашому проекті достатньо велика кількість контролерів, а, відповідно і шляхів.Тому детальний їх перелік зведемо в таблицю 2.1

Таблиця 2.1- Інформаційні зв’язки

Пефікс	**HTTP меттод**	**Шлях**	**Контроллер**
sidekiq_web_path		/sidekiq	Sidekiq::Web
api_tags_path	GET	/api/tags(.:format)	api/tags#index
new_session_path	GET	(/:locale)/session/new(.:format)	sessions#new {:locale=>/en
session_path	DELETE	(/:locale)/session(.:format)	sessions#destroy {:locale=>/en
	POST	(/:locale)/session(.:format)	sessions#create {:locale=>/en
new_password_reset_path	GET	(/:locale)/password_reset/new(.:format)	password_resets#new {:locale=>/en

Продовження таблиці 2.1

edit_password_reset_path	GET	(/:locale)/password_reset/edit(.:format)	password_resets#edit {:locale=>/en
password_reset_path	PATCH	(/:locale)/password_reset(.:format)	password_resets#update {:locale=>/en
	PUT	(/:locale)/password_reset(.:format)	password_resets#update {:locale=>/en
	POST	(/:locale)/password_reset(.:format)	password_resets#create {:locale=>/en
users_path	POST	(/:locale)/users(.:format)	users#create {:locale=>/en
new_user_path	GET	(/:locale)/users/new(.:format)	users#new {:locale=>/en
edit_user_path	GET	(/:locale)/users/:id/edit(.:format)	users#edit {:locale=>/en
user_path	PATCH	(/:locale)/users/:id(.:format)	users#update {:locale=>/en
	PUT	(/:locale)/users/:id(.:format)	users#update {:locale=>/en

Продовження таблиці 2.1

question_answers_path	GET	(/:locale)/questions/:question_id/answers(.:format)	answers#index {:locale=>/en
	POST	(/:locale)/questions/:question_id/answers(.:format)	answers#create {:locale=>/en
edit_question_answer_path	GET	(/:locale)/questions/:question_id/answers/:id/edit(.:format)	answers#edit {:locale=>/en
question_answer_path	PATCH	(/:locale)/questions/:question_id/answers/:id(.:format)	answers#update {:locale=>/en
	PUT	(/:locale)/questions/:question_id/answers/:id(.:format)	answers#update {:locale=>/en
	DELETE	(/:locale)/questions/:question_id/answers/:id(.:format)	answers#destroy {:locale=>/en
question_comments_path	POST	(/:locale)/questions/:question_id/comments(.:format)	comments#create {:locale=>/en
question_comment_path	DELETE	(/:locale)/questions/:question_id/comments/:id(.:format)	comments#destroy {:locale=>/en
questions_path	GET	(/:locale)/questions(.:format)	questions#index {:locale=>/en

Продовження таблиці 2.1

	POST	(/:locale)/questions(.:format)	questions#create {:locale=>/en
new_question_path	GET	(/:locale)/questions/new(.:format)	questions#new {:locale=>/en
edit_question_path	GET	(/:locale)/questions/:id/edit(.:format)	questions#edit {:locale=>/en
question_path	GET	(/:locale)/questions/:id(.:format)	questions#show {:locale=>/en
	PATCH	(/:locale)/questions/:id(.:format)	questions#update {:locale=>/en
	PUT	(/:locale)/questions/:id(.:format)	questions#update {:locale=>/en
	DELETE	(/:locale)/questions/:id(.:format)	questions#destroy {:locale=>/en
answer_comments_path	POST	(/:locale)/answers/:answer_id/comments(.:format)	comments#create {:locale=>/en
answer_comment_path	DELETE	(/:locale)/answers/:answer_id/comments/:id(.:format)	comments#destroy {:locale=>/en

Продовження таблиці 2.1

answers_path	GET	(/:locale)/answers(.:format)	answers#index {:locale=>/en
	POST	(/:locale)/answers(.:format)	answers#create {:locale=>/en
edit_answer_path	GET	(/:locale)/answers/:id/edit(.:format)	answers#edit {:locale=>/en
answer_path	PATCH	(/:locale)/answers/:id(.:format)	answers#update {:locale=>/en
	PUT	(/:locale)/answers/:id(.:format)	answers#update {:locale=>/en
	DELETE	(/:locale)/answers/:id(.:format)	answers#destroy {:locale=>/en
admin_users_path	GET	(/:locale)/admin/users(.:format)	admin/users#index {:locale=>/en
	POST	(/:locale)/admin/users(.:format)	admin/users#create {:locale=>/en
edit_admin_user_path	GET	(/:locale)/admin/users/:id/edit(.:format)	admin/users#edit {:locale=>/en

Продовження таблиці 2.1

admin_user_path	PATCH	(/:locale)/admin/users/:id(.:format)	admin/users#update {:locale=>/en
	PUT	(/:locale)/admin/users/:id(.:format)	admin/users#update {:locale=>/en
	DELETE	(/:locale)/admin/users/:id(.:format)	admin/users#destroy {:locale=>/en
root_path	GET	(/:locale)(.:format)	pages#index {:locale=>/en
rails_postmark_inbound_emails_path	POST	/rails/action_mailbox/postmark/inbound_emails(.:format)	action_mailbox/ingresses/postmark/inbound_emails#create
rails_relay_inbound_emails_path	POST	/rails/action_mailbox/relay/inbound_emails(.:format)	action_mailbox/ingresses/relay/inbound_emails#create
rails_sendgrid_inbound_emails_path	POST	/rails/action_mailbox/sendgrid/inbound_emails(.:format)	action_mailbox/ingresses/sendgrid/inbound_emails#create
rails_mandrill_inbound_health_check_path	GET	/rails/action_mailbox/mandrill/inbound_emails(.:format)	action_mailbox/ingresses/mandrill/inbound_emails#health_check

2.5 Написання тексту програми

При написанні програми використовувалися бібліотеки, які подано в таблиці 2.2

Таблиця 2.2 – Бібліотеки проекту

Назва	Група	Опис
Rails 6.1.3	default	Фреймворк для розробки веб-додатків на Ruby
sqlite3 1.4	default	Клієнт бази даних SQLite3 для Ruby
puma 5.0	default	Веб-сервер Ruby для використання з Rails
Webpacker 6	default	Інтеграція Webpack з Rails для керування ресурсами
bcrypt 3.1.7	default	Бібліотека для хешування паролів
activerecord-import 1.2	default	Забезпечує масове імпортування даних в ActiveRecord
blueprinter 0.25	default	Бібліотека для серіалізації об'єктів в JSON API форматі
caxlsx 3.1	default	Гем для генерації Excel-файлів з Ruby
caxlsx_rails 0.6	default	Інтеграція caxlsx з Ruby on Rails
dotenv-rails 2.7	default	Завантажує змінні середовища з файлу .env

Продовження таблиці 2.2

draper 4.0	default	Допоміжна бібліотека для декорування об'єктів в Rails
i18n-tasks 0.9	default	Інструмент для роботи з локалізацією в Rails
lokalise_rails 3	default	Інтеграція з сервісом локалізації Lokalise в Rails
pagy 5.0	default	Бібліотека для пагінації результатів в Rails
Pundit 2.1	default	Фреймворк для авторизації та контролю доступу в Rails
rails-i18n 6	default	Офіційний набір локалей для Rails
rubyXL 3.4	default	Бібліотека для роботи з Excel-файлами в Ruby
rubyzip 2	default	Бібліотека для створення та розпакування ZIP-архівів
sidekiq 6	default	Фреймворк для асинхронних завдань у Rails
valid_email2 4.0	default	Валідація електронної пошти в Rails
byebug	development	Розширений відладчик для Ruby

Продовження таблиці 2.2

faker 2	development	Генератор фейкових даних для розробки тестів
pry-rails	development	Підтримка Pry в Rails консолі
web-console 4.1.0	development	Інтерактивна консоль для веб-додатків в Rails
bullet	development	Гем для виявлення непотрібних запитів до бази даних
letter_opener	development	Відкриває електронні листи в браузері для розробки
rack-mini-profiler 2.0	development	Інструмент для аналізу продуктивності в Rails
Rubocop 1.18	development	Інструмент для автоматичної перевірки стилю коду
rubocop-i18n 3	development	Правила RuboCop для локалізації
rubocop-performance 1.11	development	Правила RuboCop для виявлення проблем продуктивності
rubocop-rails 2.11	development	Правила RuboCop для виявлення проблем в Rails
tzinfo-data	default	Дані про часові пояси для Windows

Продовження таблиці 2.2

Назва гему	Група	Опис
blueprinter	default	Бібліотека для серіалізації об'єктів в JSON API форматі
	default	Створення структур JSON за допомогою DSL в стилі Builder.
bootstrap 5.1.0	default	Фреймворк HTML, CSS і JavaScript для розробки адаптивних мобільних проектів в Інтернеті.
caxlsx	default	Гем для генерації Excel-файлів з Ruby
devise 4.6.2	default	Гнучке рішення аутентифікації для Rails з Warden.
devise-bootstrap-views 1.0	default	Стилістичне оформлення Bootstrap для Devise.

Для написання додатку використано середовище розробки RubyMine (див.рис.2.11)

RubyMine представляє собою інтегроване середовище розробки (IDE) для мови програмування Ruby. Воно надає розширену функціональність для редагування, налагодження та управління проектами на Ruby. RubyMine пропонує зручний інтерфейс та набір інструментів, спеціально розроблених для ефективної роботи з Ruby та його фреймворками, такими як Ruby on Rails.

Однією з ключових функціональних можливостей RubyMine є високопродуктивний редактор коду, який надає розширену підсвітку синтаксису, автодоповнення, перевірку наявних помилок та вбудовану документацію.

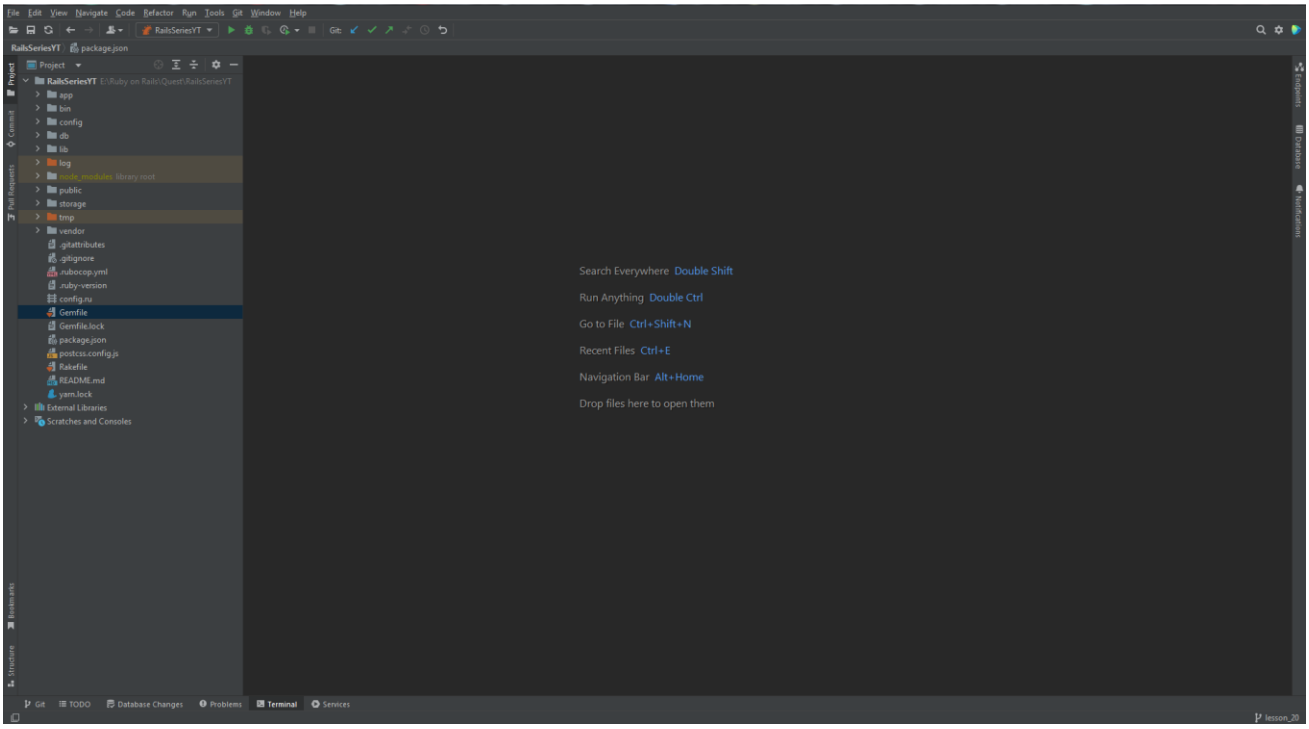

Рисунок 2.11 – Середовище розробки RubyMine

Редактор підтримує також Markdown, що дозволяє форматувати статті та документацію.

RubyMine надає розширений набір інструментів для роботи з проектами, включаючи вбудовані інструменти для управління версіями. Ці інструменти дозволяють зручно працювати з Git, SVN та іншими системами контролю версій. Крім того, IDE підтримує відладку коду, надаючи можливість крокувати по коду, встановлювати точки зупинки та аналізувати значення змінних.

Крім розширених можливостей для редагування та управління кодом, RubyMine надає інструменти для рефакторингу коду, що спрощують поліпшення структури та якості програмного коду. IDE пропонує різноманітні опції рефакторингу, такі як перейменування змінних, екстракція методів, оптимізація імпортів та багато інших.

Також варто зазначити, що RubyMine підтримує розробку веб-додатків на основі Ruby on Rails. Інтеграція з фреймворком Rails спрощує створення нових проектів, генерацію моделей та контролерів, міграції бази даних та інші аспекти розробки Ruby on Rails. Завдяки цьому, розробники можуть зручно працювати з усіма аспектами розробки Ruby on Rails в межах одного IDE.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Аналіз нормативно-правової бази

У веб-середовищі, де взаємодіють користувачі, зберігаються та обмінюються інформацією, надання належного захисту інформації та персональних даних користувачів відіграє важливу роль. В Україні ці питання регулюються рядом законодавчих актів, зокрема Закону України "Про захист інформації в інформаційно-комунікаційних системах" та Закону України "Про захист персональних даних".

Закон України "Про захист інформації в інформаційно-комунікаційних системах" має на меті забезпечення цілісності, конфіденційності, доступності та надійності інформації в інформаційно-комунікаційних системах. Він встановлює вимоги до захисту інформації, включаючи обмеження доступу до неї, забезпечення конфіденційності та захист від несанкціонованого доступу, а також застереження щодо цілісності інформації.

Враховуючи вимоги Закону "Про захист інформації в інформаційно-комунікаційних системах", веб-сайт системи питань та відповідей повинен забезпечувати належний рівень захисту інформації шляхом використання сучасних технічних засобів, таких як шифрування даних, механізми аутентифікації та контролю доступу. Потрібно забезпечити конфіденційність обміну даними між користувачами та системою, а також запобігти можливим атакам на систему, що можуть призвести до несанкціонованого доступу до інформації.

Додатково, важливим є дотримання Закону України "Про захист персональних даних". Цей закон визначає правові принципи збору, зберігання, використання та захисту персональних даних громадян. Він передбачає, що збір та обробка персональних даних можуть здійснюватись лише за згодою суб'єкта персональних даних або на підставі закону. Також

передбачається обов'язкова інформація користувачам щодо цілей збору та обробки їх персональних даних, а також права суб'єктів персональних даних на доступ до своїх даних та внесення змін до них.

НД ТЗІ 2.5-010-03 "Вимоги до захисту інформації WEB-сторінки від несанкціонованого доступу" є ще одним важливим документом, який слід враховувати при розробці та експлуатації веб-сайту системи питань та відповідей. Цей документ встановлює конкретні вимоги до захисту інформації, яка міститься на WEB-сторінках, з метою запобігання несанкціонованому доступу до неї.

Згідно з НД ТЗІ 2.5-010-03, веб-сторінки повинні бути захищені від несанкціонованого доступу шляхом використання різноманітних технічних, організаційних та адміністративних заходів. Основні вимоги, які встановлює цей документ, включають:

НД ТЗІ 2.5-010-03 містить такі основні вимоги до захисту інформації на WEB-сторінках:

- Аутентифікація та авторизація: Веб-сторінка повинна використовувати механізми аутентифікації, щоб перевірити ідентифікацію користувачів та перевірити їх права доступу до різних функцій та ресурсів. Зазвичай це досягається за допомогою унікальних ідентифікаторів користувачів і паролів, а також може включати двофакторну аутентифікацію або використання сертифікатів.
- Шифрування: Для забезпечення конфіденційності передачі даних між користувачем та веб-сторінкою, важливо використовувати шифрування. Протоколи шифрування, такі як SSL (Secure Sockets Layer) або його наступник TLS (Transport Layer Security), дозволяють захищено передавати інформацію шляхом шифрування даних під час їх трансляції через мережу.
- Захист від SQL-ін'єкцій та інших вразливостей: Важливо розробляти веб-сторінку з урахуванням заходів захисту від вразливостей, таких як SQL-ін'єкція, кросс-сайтовий скриптінг та інші типи атак. Це може включати правильну обробку та валідацію вхідних даних, використання

параметризованих запитів до бази даних та застосування механізмів фільтрації та санітизації вхідних даних.

- Моніторинг та журналювання: Веб-сторінка повинна мати механізми моніторингу та журналювання подій, що відбуваються на ній. Це дозволяє виявляти та реагувати на можливі атаки або спроби несанкціонованого доступу, а також встановлювати причини і наслідки подій, що стосуються безпеки.
- Резервне копіювання та відновлення: Регулярне резервне копіювання даних веб-сторінки та виконання планів відновлення дозволяють забезпечити можливість відновлення системи в разі втрати даних або атаки.

НД ТЗІ 2.5-010-03 встановлює ці вимоги з метою запобігання несанкціонованому доступу до WEB-сторінки та забезпечення належного рівня захисту інформації.

Дотримання цих вимог є важливим для забезпечення безпеки та конфіденційності інформації, яка обмінюється на WEB-сторінці.

3.2 Розробка політики безпеки

Політика безпеки - це набір правил, процедур і налаштувань, розроблених для забезпечення безпеки веб-сайту та захисту від потенційних загроз. Ця політика визначає підходи, які веб-сайт використовує для захисту від несанкціонованого доступу, витоку даних, вразливостей програмного забезпечення, зловживання привілеями тощо.

Політика безпеки веб-сайту є комплексним підходом до забезпечення безпеки і захисту інформації на веб-сайті. Вона включає такі аспекти, як аутентифікація і авторизація користувачів, захист даних, захист від зловмисних атак, оновлення програмного забезпечення та планування резервного копіювання.

Цей підхід дозволяє веб-сайтам забезпечувати високий рівень безпеки, запобігаючи несанкціонованому доступу до конфіденційної інформації та

зберігаючи надійність та доступність веб-сайту для користувачів. Важливо постійно оновлювати та адаптувати політику безпеки веб-сайту до нових загроз та використовувати передові технології та методи для забезпечення безпеки веб-сайту.

Цілі політики безпеки веб-додатку включають в себе широкий спектр аспектів, які мають на меті забезпечити безпеку, конфіденційність, цілісність та доступність даних. Основні цілі політики безпеки можуть бути наступними:

- Захист від несанкціонованого доступу: Політика безпеки повинна містити заходи, що забезпечують захист від несанкціонованого доступу до додатку та його ресурсів. Це включає встановлення міцних механізмів аутентифікації, контроль доступу, захист мережевого зв'язку та ідентифікацію користувачів.

- Захист від втрати даних: Політика безпеки має на меті захистити дані від випадкової або навмисної втрати. Це включає резервне копіювання та відновлення даних, застосування механізмів реплікації, шифрування даних та забезпечення фізичної безпеки серверів та сховищ.

- Забезпечення конфіденційності даних: Політика безпеки повинна забезпечувати конфіденційність даних, забезпечуючи захист від несанкціонованого доступу до чутливої інформації. Це можна досягти за допомогою шифрування даних, використанням безпечних протоколів зв'язку, контролем доступу та обмеженням доступу до конфіденційної інформації лише необхідним користувачам.

- Захист від атак та вразливостей: Політика безпеки має на меті захистити додаток від вразливостей та різних видів атак, таких як Cross-Site Scripting (XSS), SQL-ін'єкція, міжсайтова поділена автентифікація (Session Hijacking) та інші. Це включає валідацію та екранування вхідних даних, використання безпечних бібліотек та фреймворків, регулярні оновлення та патчі.

- Забезпечення доступності: Політика безпеки також має забезпечити доступність додатку для легітимних користувачів. Це включає захист від Denial of Service (DoS) атак, забезпечення масштабованості системи, відновлення після вторгнень та забезпечення належної продуктивності.

- Дотримання законодавства та стандартів безпеки: Політика безпеки повинна враховувати вимоги законодавства та стандартів безпеки, які відповідають конкретній сфері діяльності додатку. Наприклад, це може бути вимога до захисту персональних даних (GDPR), стандарти безпеки платіжних карт (PCI DSS) тощо.

Цілі політики безпеки спрямовані на створення безпечного, надійного та відповідного законодавству веб-додатку на Ruby on Rails 6. Дотримання цих цілей допоможе забезпечити захищеність даних та зберегти довіру користувачів у вашому додатку.

Загальні вимоги політики безпеки є важливими для забезпечення захищеності веб-додатків. Врахування цих вимог допоможе створити міцну та надійну політику безпеки. Ось кілька загальних вимог, які можна врахувати:

1) Ідентифікація та аутентифікація:

а) Встановлення міцних механізмів аутентифікації для перевірки особистості користувачів.

б) Вимога сильних паролів та регулярна зміна паролів.

в) Реалізація двофакторної аутентифікації для додаткового рівня безпеки.

2) Керування доступом:

а) Встановлення різних рівнів доступу до функціональності та даних в залежності від ролей користувачів.

б) Реалізація механізмів контролю доступу, таких як ролева модель або політики засновані на ролі.

3) Захист від атак:

а) Запобігання атакам типу Cross-Site Scripting (XSS) шляхом екранування вхідних даних та валідації коректності введених даних.

б) Захист від атак типу SQL-ін'єкція шляхом використання параметризованих запитів та фільтрації вхідних даних.

в) Захист від атак типу Cross-Site Request Forgery (CSRF) за допомогою використання токенів аутентифікації та перевірки джерела запиту.

4) Захист від витоку інформації:

а) Використання шифрування для захисту конфіденційних даних під час зберігання та передачі.

б) Мінімізація збереження конфіденційної інформації та обмеження доступу до неї лише необхідним користувачам.

5) Моніторинг та реагування на інциденти:

а) Регулярний моніторинг додатку та журналювання подій для виявлення підозрілої активності.

б) Визначення процедур реагування на інциденти безпеки, включаючи швидку реакцію, розслідування та відновлення після вторгнень.

6) Оновлення та патчі:

а) Регулярне оновлення фреймворків, бібліотек та інших компонентів додатку для виправлення відомих вразливостей.

б) Встановлення патчів безпеки негайно після їх випуску для усунення нових вразливостей.

7) Навчання та свідомість:

а) Навчання розробників та користувачів щодо безпечних практик розробки та використання додатку.

б) Свідомість про загрози безпеки та практики їх запобігання через документацію, навчальні матеріали та інформаційні повідомлення.

Загальні вимоги політики безпеки допомагають створити систематичний та комплексний підхід до безпеки веб-додатків на Ruby on Rails 6. Врахування цих вимог підвищить безпеку вашого додатку та захистить його від різних загроз.

Реалізація політики безпеки вимагає комплексної системи захисту інформації, яка включає в себе різні компоненти та процеси. Ось деякі ключові аспекти, що варто врахувати при реалізації політики безпеки:

- Аналіз ризиків: Перший крок у реалізації політики безпеки - це проведення аналізу ризиків. Це допомагає ідентифікувати потенційні загрози та вразливості в системі, оцінити їх вплив та ймовірність виникнення. Аналіз ризиків допоможе визначити пріоритети та встановити необхідні заходи безпеки.

- Фізична безпека: Для захисту інформації важливо забезпечити фізичну безпеку обладнання та інфраструктури. Це може включати захист приміщень, контроль доступу, використання відеоспостереження та захисту від пожежі.

- Захист мережі: Забезпечення безпеки мережі є необхідним компонентом системи захисту інформації. Це включає налаштування брандмауерів, захист від вторгнень (Intrusion Detection/Prevention Systems), мережевий моніторинг та шифрування мережевого зв'язку.

- Управління ідентифікацією та доступом: Реалізація політики безпеки передбачає ефективне управління ідентифікацією користувачів та контроль доступу до системи та даних. Це може включати використання

сильних механізмів аутентифікації, ролеву модель доступу, двофакторну аутентифікацію та аудит доступу.

- Захист даних: Захист конфіденційності, цілісності та доступності даних є критично важливим аспектом політики безпеки. Він може включати шифрування даних, резервне копіювання та відновлення, механізми контролю цілісності даних та захист від витоку інформації.

- Постійний моніторинг та оновлення: Реалізація політики безпеки вимагає постійного моніторингу системи, виявлення підозрілої активності, подій та інцидентів безпеки. Регулярне оновлення програмного забезпечення, бібліотек та патчів є також необхідною складовою політики безпеки.

- Навчання та свідомість: Для ефективної реалізації політики безпеки необхідно навчати персонал, користувачів та стейкхолдерів щодо безпечних практик. Це може включати проведення тренінгів, свідомість про загрози безпеки, поширення інформації про нові вразливості та методи атак.

Реалізація політики безпеки вимагає комплексного підходу, враховуючи різні аспекти захисту інформації. Комбінація фізичної безпеки, захисту мережі, управління ідентифікацією та доступом, захисту даних, постійного моніторингу та навчання персоналу допоможе створити надійну систему захисту інформації в вашому веб-додатку.

Нормативно-правові заходи захисту інформації в Україні відіграють важливу роль у забезпеченні безпеки і конфіденційності даних. Основними законодавчими актами є:

- Закон України "Про захист інформації в інформаційно-комунікаційних системах" (Закон про ЗІКС). Цей закон був прийнятий з метою забезпечення ефективного захисту інформації, яка обробляється та передається в інформаційно-комунікаційних системах.

- Закон України "Про захист персональних даних" від 1 червня 2010 року. Цей закон встановлює правила збирання, обробки та зберігання персональних даних, а також визначає права суб'єктів даних та обов'язки операторів.

- Закон України "Про інформацію" від 2 жовтня 1992 року. Цей закон визначає загальні принципи захисту інформації, включаючи вимоги до забезпечення конфіденційності, цілісності та доступності інформації.

- Закон України "Про електронний документообіг" від 22 вересня 2005 року. Цей закон регулює електронний документообіг та встановлює вимоги до забезпечення безпеки електронних документів та електронних підписів.

- Закон України "Про кібербезпеку" від 5 липня 2017 року. Цей закон визначає основні принципи та заходи забезпечення кібербезпеки в Україні, включаючи захист інформаційних систем від кібератак та встановлення вимог до організацій, які займаються обробкою інформації.

Крім цих законодавчих актів, існують також нормативні документи, які регулюють окремі аспекти захисту інформації. Наприклад:

- Накази та розпорядження Державного агентства з питань електронного урядування та інших відповідних органів щодо впровадження технічних засобів захисту інформації, вимог до безпеки мереж та систем.

- Міжнародні стандарти, такі як ISO/IEC 27001, ISO/IEC 27002, які надають рекомендації та вимоги щодо управління безпекою інформації.

Дотримання нормативно-правових вимог у сфері захисту інформації в Україні є обов'язковим для всіх організацій та установ, які займаються обробкою та зберіганням інформації. Виконання цих вимог допомагає забезпечити конфіденційність, цілісність та доступність даних, а також запобігти можливим вразливостям та кібератакам.

Організаційні заходи захисту інформації є важливою складовою політики безпеки. Вона застосовується для забезпечення конфіденційності, цілісності та доступності інформації в організації. Ці заходи охоплюють широкий спектр дій, спрямованих на попередження загроз та мінімізацію ризиків для інформаційних ресурсів. Нижче наведено кілька основних організаційних заходів захисту інформації:

- Розробка політики безпеки: Організація повинна розробити документ, який визначає загальні принципи та вимоги щодо захисту інформації. Цей документ повинен включати правила щодо обробки, зберігання, передачі та використання інформації, а також встановлювати відповідальності та ролі працівників у забезпеченні безпеки.

- Класифікація інформації: Організація повинна класифікувати інформацію залежно від її значущості та ступеня конфіденційності. Це допоможе визначити рівень захисту, який необхідно застосувати до кожного типу інформації.

- Управління доступом: Організація повинна встановити механізми контролю доступу до інформаційних ресурсів. Це включає використання паролів, обмеження прав доступу до конфіденційної інформації, аутентифікацію та авторизацію користувачів.

- Навчання та свідомість персоналу: Організація повинна забезпечити навчання та підвищення свідомості свого персоналу щодо важливості безпеки інформації. Це може включати проведення тренінгів, навчальних програм та інформаційних кампаній з безпеки.

- Резервне копіювання та відновлення: Організація повинна регулярно робити резервні копії важливої інформації та матеріалів і мати план відновлення в разі аварійного випадку. Це дозволить забезпечити доступність інформації навіть у разі втрати або пошкодження даних.

- Моніторинг та аудит безпеки: Організація повинна встановити системи моніторингу та аудиту, які дозволять виявляти потенційні загрози та вразливості, а також виявляти неправильну поведінку або недостовірну діяльність. Це допоможе вчасно реагувати на можливі загрози та викрити можливі порушення безпеки.

Ці організаційні заходи захисту інформації є важливими для забезпечення безпеки інформаційних ресурсів організації. Вони повинні бути впроваджені та регулярно переглядатися для впевненості у високому рівні захисту інформації та запобігання можливим загрозам.

Інженерно-технічні засоби захисту інформації є важливою складовою політики безпеки, які включають різноманітні технічні рішення та засоби, які застосовуються для забезпечення цілісності, конфіденційності та доступності інформації. Вони допомагають усувати загрози та мінімізувати ризики для інформаційних ресурсів організації. Основні інженерно-технічні засоби захисту інформації включають:

- Безпека мережі: Використання фаерволів, інтрузійних виявлення та запобігання (IDS/IPS), віртуальних приватних мереж (VPN) та інших мережевих засобів для захисту мережевого середовища від несанкціонованого доступу та зловмисних атак.

- Криптографічні засоби: Використання алгоритмів шифрування та підпису для захисту конфіденційності та цілісності інформації під час її передачі та зберігання. Це включає використання SSL/TLS протоколів для захищеної передачі даних через мережу та використання криптографічних ключів.

- Системи ідентифікації та автентифікації: Використання систем, які перевіряють ідентифікацію користувачів та їх автентифікацію перед наданням доступу до систем та ресурсів. Це можуть бути системи одноразових паролів (OTP), біометричні системи, двофакторна аутентифікація та інші засоби.

- Фізична безпека: Забезпечення фізичної безпеки серверних кімнат, дата-центрів та інших приміщень, де знаходяться інформаційні системи та обладнання. Це включає контроль доступу до приміщень, використання систем відеоспостереження, систем захисту від проникнення та інші фізичні заходи.

- Захист зовнішніх пристроїв: Використання засобів захисту для зовнішніх пристроїв, таких як USB-накопичувачі, зовнішні жорсткі диски та інші зовнішні пристрої зберігання даних. Це може включати шифрування даних на цих пристроях та використання політик безпеки щодо їх використання.

- Системи моніторингу та логування: Встановлення систем, які ведуть моніторинг активності користувачів та подій у системі, а також зберігають логи для подальшого аналізу та виявлення вразливостей та зловмисних дій.

Використання інженерно-технічних засобів захисту інформації є необхідним для створення комплексної системи безпеки, яка забезпечує високий рівень захисту інформації в організації. При впровадженні таких засобів важливо враховувати специфіку діяльності організації, її ризики та потреби у забезпеченні безпеки.

Керування доступом є важливою складовою політики безпеки інформації. Організації повинні використовувати різноманітні організаційні заходи для ефективного керування доступом до систем, даних та ресурсів. Основною метою таких заходів є забезпечення тільки необхідного рівня доступу для користувачів залежно від їх ролей, відповідальностей та потреб.

Основні організаційні заходи щодо керування доступом включають:

- Політика доступу: Організація повинна встановити політику доступу, яка визначає правила та процедури для керування доступом до систем та ресурсів. Політика повинна враховувати потреби організації, ризики та вимоги відповідних нормативно-правових актів.

- Ідентифікація та аутентифікація: Організація повинна мати механізми для ідентифікації користувачів і перевірки їх автентичності перед наданням доступу. Це може включати використання унікальних імен користувачів, паролів, біометричних даних, токенів або інших методів аутентифікації.

- Управління ролями та привілеями: Організація повинна використовувати систему управління ролями та привілеями, щоб керувати доступом користувачів до різних ресурсів на основі їхніх ролей та відповідальностей. Це дозволяє обмежити доступ до конфіденційної інформації тільки тим користувачам, які мають відповідні права.

- Аудит та моніторинг доступу: Організація повинна встановити систему аудиту та моніторингу доступу, щоб відстежувати активності користувачів, перевіряти виконання політик безпеки та виявляти недоречний або підозрілий доступ. Це допомагає виявити можливі порушення безпеки та забезпечити відповідну реакцію на них.

- Освіта та навчання: Організація повинна забезпечити своїх співробітників навчальними програмами та інформаційними матеріалами щодо безпеки інформації та правил користування системами. Регулярні навчання та підвищення свідомості щодо безпеки допомагають запобігти недбалому використанню або неналежному керуванню доступом.

- Управління життєвим циклом доступу: Організація повинна забезпечити ефективне управління життєвим циклом доступу, включаючи створення, зміну та припинення доступу до систем та ресурсів. Це може включати процеси огляду та оновлення прав доступу згідно зі змінами у ролях користувачів або їхньої зайнятості в організації.

Організаційні заходи щодо керування доступом відіграють важливу роль у забезпеченні безпеки інформації. Вони допомагають уникнути несанкціонованого доступу до систем, мінімізувати ризик втрати чутливих даних та зберегти довіру відповідних зацікавлених сторін.

Організаційні заходи щодо забезпечення цілісності інформації відіграють важливу роль у забезпеченні безпеки інформаційних ресурсів та захисту від несанкціонованих змін, втрати або пошкодження даних. Ці заходи спрямовані на запобігання несанкціонованій модифікації або втраті даних, а також на забезпечення їхньої цілісності протягом усього життєвого циклу інформації.

Основні організаційні заходи щодо забезпечення цілісності інформації включають:

- Розробка політик та процедур: Організація повинна розробити політики та процедури, які визначають стандарти та вимоги щодо забезпечення цілісності інформації. Ці документи повинні включати визначення процедур контролю доступу, резервного копіювання та відновлення даних, а також механізми перевірки цілісності даних.

- Управління правами доступу: Організація повинна забезпечити ефективне управління правами доступу до інформаційних ресурсів. Це включає надання користувачам лише необхідного рівня доступу до даних та ресурсів, а також використання механізмів контролю доступу, таких як ролі, привілеї та обмеження доступу на основі потреб та відповідальностей користувачів.

- Моніторинг та аудит цілісності даних: Організація повинна використовувати механізми моніторингу та аудиту для перевірки цілісності даних. Це включає виявлення та реагування на будь-які незвичайні або підозрілі зміни в інформаційних ресурсах. Моніторинг може включати в себе використання систем журналювання подій та механізмів виявлення вторгнень.

- Забезпечення фізичної безпеки: Організація повинна забезпечити фізичну безпеку інформаційних ресурсів. Це включає контроль доступу до приміщень, де зберігаються сервери та інші інформаційні пристрої, застосування систем відеоспостереження, а також резервне копіювання та зберігання даних у безпечних місцях.

- Навчання та освіта персоналу: Організація повинна забезпечити навчання та освіту свого персоналу щодо правил та процедур забезпечення цілісності інформації. Персонал повинен бути ознайомлений з потенційними загрозами цілісності та навичками для виявлення та запобігання несанкціонованим змінам даних.

- Захист від зовнішніх загроз: Організація повинна використовувати заходи для захисту від зовнішніх загроз, таких як зломи, вторгнення або вірусні атаки. Це може включати застосування брандмауерів, антивірусного програмного забезпечення та регулярне оновлення програмного забезпечення для попередження вразливостей.

Забезпечення цілісності інформації вимагає комплексного підходу, включаючи як організаційні, так і технічні заходи. Комбінація цих заходів допоможе організації запобігти несанкціонованій модифікації, втраті або пошкодженню даних, забезпечити надійність та довіру відповідних інформаційних ресурсів.

3.3 Технічний захист інформації

Хешування паролів є важливою складовою безпеки в сучасних системах комп'ютерної безпеки. В основі хешування паролів лежить процес перетворення введеного паролю у складну послідовність символів, яка називається хешем. Хеш є результатом використання хеш-функції, алгоритму, який приймає вхідний пароль і генерує вихідний хеш.

Однією з основних вимог до хеш-функцій є незворотність: необхідно, щоб було практично неможливо відновити вхідний пароль за його хешем. Це означає, що хеш-функція має бути односторонньою, тобто легко обчислюється для будь-якого паролю, але дуже складно знайти вихідний пароль за його хешем.

Іншою важливою властивістю хеш-функцій є стійкість до колізій. Колізія виникає, коли два різних вхідних паролів генерують однаковий хеш. Хеш-функція повинна бути такою, щоб ймовірність виникнення колізій була надзвичайно низькою. Це забезпечує, що навіть якщо зловмисник отримає доступ до хешів паролів, він не зможе ефективно знайти відповідні паролі, що відповідають цим хешам.

Для забезпечення більшої безпеки паролів, перед хешуванням до вхідного паролю додається сіль (salt). Сіль - це випадкова додаткова інформація, яка додається до паролю перед хешуванням. Використання солі унеможливлює використання попередньо згенерованих таблиць радянь (rainbow tables), які використовуються для швидкого підбору паролів за їх хешами.

Хешування паролів має велике значення для безпеки веб-систем. При реєстрації нового користувача його пароль зазвичай хешується і зберігається у базі даних. При наступній авторизації користувача система порівнює хеш введеного паролю зі збереженим хешем. Якщо вони співпадають, то пароль вважається вірним.

Хешування паролів є ефективним методом забезпечення безпеки, оскільки воно ускладнює завдання зловмисникам при отриманні доступу до паролів. Навіть якщо база даних з хешами паролів потрапить у руки зловмисника, важко відновити вихідні паролі без значних обчислювальних зусиль. Тому рекомендується використовувати сильні хеш-функції та додавати до паролів випадкову сіль для підвищення рівня безпеки.

В Ruby on Rails 6 для хешування паролів за замовчуванням використовується алгоритм bcrypt. Bcrypt є одним з найбільш надійних алгоритмів хешування паролів і забезпечує високий рівень безпеки.

У Ruby on Rails bcrypt використовується за допомогою гему bcrypt. Цей гем забезпечує зручний і простий спосіб хешування паролів і порівняння хешів для авторизації користувачів.

При реєстрації нового користувача пароль перед хешуванням може бути збережений у змінній password моделі користувача. Для хешування

цього паролю необхідно викликати метод has_secure_password у моделі користувача. Цей метод додає необхідну функціональність для роботи з паролем, включаючи генерацію хеша та перевірку пароля.

При створенні об'єкта користувача з паролем, гем bcrypt автоматично хешує пароль та зберігає його у вигляді хеш-строки. Цей хеш потім можна зберегти у базі даних.

При наступній авторизації користувача, коли він вводить свій пароль, застосовується метод authenticate на об'єкті користувача. Цей метод порівнює введений пароль збереженому хешу пароля, використовуючи bcrypt. Якщо пароль співпадає, метод authenticate повертає об'єкт користувача, що дозволяє робити подальшу обробку аутентифікації.

Однією з переваг bcrypt є його обчислювальна витривалість. Він використовує адаптивну хеш-функцію, яка виконує багато ітерацій хешування, що робить його вразливим до атак перебору паролів. Тим самим, bcrypt ускладнює завдання зловмисникам, які намагаються відновити оригінальний пароль за його хешем.

Важливою додатковою функціональністю, яку надає bcrypt у Ruby on Rails, є автоматична генерація випадкової солі для кожного пароля. Сіль додається до паролю перед хешуванням і зберігається разом з хешем пароля. Це підвищує безпеку паролів, оскільки унеможливлює використання попередньо згенерованих таблиць радянь для швидкого підбору паролів.

Загалом, використання bcrypt у Ruby on Rails дозволяє забезпечити надійний рівень безпеки для хешування паролів користувачів. Цей алгоритм, разом з функціональністю, що надає гем bcrypt, забезпечує ефективну та безпечну обробку паролів у веб-додатках на основі Ruby on Rails.

Розробка механізмів валідації.

Розробка механізмів валідації даних є важливою складовою безпеки веб-додатків. Цей процес включає перевірку та фільтрацію введених користувачем даних, щоб запобігти вразливостям, таким як впровадження зловмисницького коду, втручання у базу даних та інші атаки на веб-додаток.

Основна мета механізмів валідації даних - забезпечити, щоб введені користувачем дані задовольняли певним правилам та форматам. Ось кілька важливих механізмів валідації даних для запобігання вразливостям веб-додатку:

- Перевірка на стороні клієнта: Валідація виконується за допомогою JavaScript та HTML5. Вона може включати перевірку обов'язкових полів, форматів даних (наприклад, електронної пошти, номера телефону) та довжини введених даних. Це дозволяє забезпечити швидку зворотну зв'язок з користувачем щодо некоректно введених даних, не надсилати їх на сервер та полегшити роботу з обробкою помилок.

- Перевірка на стороні сервера: Валідація на сервері є обов'язковою, оскільки валідація на стороні клієнта може бути обхідною. Вона включає перевірку коректності та цілісності введених даних перед збереженням або обробкою на сервері. Наприклад, перевірка наявності обов'язкових полів, обмеження на довжину даних, перевірка унікальності значень у базі даних та фільтрація зловмисницького коду (наприклад, SQL-ін'єкцій, скриптів Cross-Site Scripting).

- Валідація типів даних: Крім перевірки правильності введених даних, важливо переконатися, що дані відповідають очікуваним типам. Наприклад, перевірка, чи введене значення числове, дата, булеве значення тощо. Це допомагає уникнути помилок обробки даних та забезпечити правильність роботи додатка.

- Захист від перевантажень та атак на сервер: Важливо встановити обмеження на обробку даних, які надсилаються користувачем, щоб запобігти перевантаженням та атакам на сервер. Наприклад, обмеження на розмір завантажуваних файлів або кількість запитів, що можуть бути виконані в певний проміжок часу.

- Оновлення та безпека бібліотек: Важливо постійно відстежувати оновлення та безпекові патчі для використовуваних бібліотек та

фреймворків. Це дозволяє уникнути використання застарілих версій з відомими вразливостями та забезпечити безпеку веб-додатку.

Усі ці механізми валідації даних використовуються для забезпечення безпеки веб-додатку та запобігання вразливостям. Вони мають бути належно налаштовані та використовуватися в поєднанні з іншими заходами безпеки, такими як автентифікація та авторизація, щоб створити надійну та безпечну систему.

У Ruby on Rails механізми валідації даних грають важливу роль у забезпеченні правильності та цілісності даних, що вводяться користувачами. Вони дозволяють перевірити, чи введені дані відповідають певним правилам та критеріям, і забезпечити безпеку та надійність веб-додатків. Давайте розглянемо, як працюють механізми валідації даних у Ruby on Rails.

У Ruby on Rails валідація даних здійснюється на рівні моделей, що дозволяє централізовано контролювати та обробляти дані, перед тим як вони будуть збережені у базі даних. Для валідації полів моделі можна використовувати різні типи валідаторів, що надаються самим Ruby on Rails або додатковими гемами.

Один з найпоширеніших валідаторів у Ruby on Rails - presence. Цей валідатор перевіряє, чи поле не є порожнім (nil) або чи не містить лише пробіли. Наприклад, для перевірки обов'язковості поля name у моделі User, можна використати наступний код:

Лістинг 3.1

```
class User < ApplicationRecord
  validates :name, presence: true
end
```

Крім presence, Ruby on Rails надає інші валідатори, такі як length, numericality, format, uniqueness та багато інших. За допомогою цих валідаторів можна встановлювати обмеження на довжину полів, перевіряти числові значення, форматувати дані відповідно до заданого шаблону та перевіряти унікальність значень у базі даних.

Наприклад, для валідації поля email у моделі User, щоб перевірити, чи має воно правильний формат та чи є унікальним, можна використати наступний код:

Лістинг 3.2

```
class User < ApplicationRecord
  validates :email, presence: true, format: { with: URI::MailTo::EMAIL_REGEXP }, uniqueness: true
end
```

Крім вбудованих валідаторів, у Ruby on Rails також можна створювати власні валідатори. Це дозволяє розширити можливості валідації та виконувати більш специфічні перевірки.

При виконанні операції збереження моделі, Rails автоматично перевіряє валідність моделі за допомогою визначених валідаторів. Якщо дані не відповідають валідації, модель не зберігається, а замість цього генерується помилка, яку можна обробити у контролері та відобразити користувачеві.

Валідація даних у Ruby on Rails дозволяє забезпечити надійність, безпеку та цілісність даних у веб-додатках. Використовуючи вбудовані валідатори та можливість створювати власні, розробники можуть контролювати та перевіряти дані перед їх збереженням, забезпечуючи надійну та безпечну роботу додатків.

Робота з сесіями в веб-додатках є важливим аспектом, але вона також відкриває можливості для різних загроз безпеки. Належна увага до безпеки сесій є обов'язковою для запобігання атакам і збереження конфіденційності даних користувачів. Оглянемо деякі з основних загроз безпеці, пов'язаних з роботою з сесіями, а також заходи, які можна вжити для їх усунення.

- Перехоплення сесій (Session Hijacking): Це загроза, при якій зловмисник отримує доступ до ідентифікатора сесії користувача і використовує його для підміни справжнього користувача. Для запобігання таким атакам, рекомендується використовувати механізми шифрування, такі як HTTPS, для передачі ідентифікаторів сесій. Також можна використовувати

техніки, такі як "затвердження" (session fixation), щоб забезпечити, що ідентифікатор сесії змінюється після аутентифікації користувача.

- Витік інформації (Information Leakage): Некоректна конфігурація або обробка помилок може призвести до витоку конфіденційної інформації через сесії. Розробники повинні перевірити, що конфіденційна інформація, така як паролі або персональні дані, не зберігається у сесійних змінних або в інших незахищених місцях. Крім того, важливо належним чином обробляти помилки, щоб не розкривати надмірну інформацію про структуру або конфігурацію додатка.

- Фіксація сесій (Session Fixation): Ця загроза виникає, коли зловмисник змушує користувача використовувати ідентифікатор сесії, який він контролює. Це може бути досягнуто шляхом передачі ідентифікатора сесії через посилання або параметри URL. Щоб уникнути фіксації сесій, рекомендується генерувати новий ідентифікатор сесії після кожної аутентифікації або переходу на критичні сторінки.

- Крадіжка сесій (Session Theft): Ця загроза полягає в тому, що зловмисник краде ідентифікатор сесії користувача для отримання несанкціонованого доступу. Для запобігання крадіжки сесій необхідно застосовувати механізми, такі як CSRF-токени (Cross-Site Request Forgery), які додаються до кожного запиту та перевіряються на сервері. Також варто регулярно оновлювати ідентифікатори сесій для ускладнення процесу крадіжки.

- Збільшення привілеїв (Privilege Escalation): Ця загроза виникає, коли зловмисник намагається змінити свої привілеї в системі, щоб отримати більше прав доступу. Для запобігання збільшенню привілеїв, необхідно належним чином налаштувати ролі та дозволи користувачів, а також здійснювати перевірку привілеїв перед виконанням критичних операцій.

Усі ці загрози можуть бути усунені або пом'якшені шляхом належного застосування безпечних практик у розробці веб-додатків. До таких практик

належать використання шифрування для передачі сесійних ідентифікаторів, належна конфігурація серверів, обмеження доступу до сесій тільки з надійних джерел, застосування CSRF-токенів, регулярне оновлення ідентифікаторів сесій, належна обробка помилок та контроль доступу користувачів.

Загальна ідея полягає в тому, щоб розробникам забезпечити належну увагу до безпеки при роботі з сесіями та застосувати рекомендовані практики безпеки для усунення загроз і збереження надійності та конфіденційності даних користувачів.

Ruby on Rails надає ряд заходів для усунення загроз безпеці при роботі з сесіями. Ось деякі з них:

- Використання безпечного сховища сесій: Rails має підтримку різних сховищ для збереження сесій, таких як cookie-сесії, база даних, кешування та зовнішні сховища, наприклад, Redis. Рекомендується використовувати безпечні сховища, які надійно зберігають сесійні дані та застосовують механізми шифрування для забезпечення конфіденційності.

- Захист ідентифікатора сесії: Важливо захистити ідентифікатор сесії від перехоплення та зловмисницьких атак. Rails використовує механізм генерації випадкових ідентифікаторів сесій та автоматично оновлює їх після аутентифікації або підвищення привілеїв користувача.

- Використання HTTPS: Для захисту передачі сесійних ідентифікаторів та інших конфіденційних даних рекомендується використовувати протокол HTTPS з встановленим SSL-сертифікатом. Це забезпечить шифрування даних між клієнтом і сервером, ускладнюючи можливість перехоплення сесійних ідентифікаторів.

- Валідація сесій: Перед використанням сесійних даних слід здійснювати перевірку їх валідності. Rails надає можливість валідувати сесійні дані, зокрема перевіряти цілісність та відповідність очікуваним форматам.

- Захист від атак типу Cross-Site Request Forgery (CSRF) та Cross-Site Scripting (XSS): Rails має вбудовану захист від XSS-атак та CSRF-атак. Однак, розробникам слід перевірити, що ці захисти належним чином використовуються у веб-додатку. Використання методу protect_from_forgery у контролерах допомагає захистити додаток від CSRF-атак.

- Налаштування параметрів сесії: Rails дозволяє налаштовувати параметри сесій, такі як термін дії сесії, довжина ідентифікатора сесії тощо. Розробники повинні належним чином налаштувати ці параметри з урахуванням потреб безпеки та функціональності додатку.

Ці заходи допоможуть усунути загрози безпеці при роботі з сесіями в Ruby on Rails. Проте, безпека є постійним процесом, тому розробники повинні слідкувати за новими загрозами та рекомендаціями щодо безпеки та вносити відповідні зміни в свої додатки.

Ruby on Rails, як і багато інших веб-фреймворків, піддається певним вразливостям, зокрема вразливостям, пов'язаним з Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF). Ось огляд цих вразливостей та кроки, які можна підприйняти для їх запобігання.

XSS (Cross-Site Scripting): XSS є типом атаки, при якій зловмисник вводить зловісний скрипт на веб-сторінку, що відображається іншим користувачам. Це може призвести до виконання небажаного коду в браузері користувачів і порушення безпеки їх даних. Для запобігання XSS-атак слід виконувати наступні заходи:

- Екранування вхідних даних: Всі дані, які потрапляють на веб-сторінку, повинні бути екрановані перед відображенням. Rails робить це автоматично для значень, які виводяться через помічники вигляду, таких як h або html_escape.

- Використання безпечних помічників вигляду: Rails надає безпечні помічники, які автоматично екранують дані перед відображенням.

Наприклад, використовуйте raw для виведення небезпечного HTML-коду тільки тоді, коли ви впевнені в його безпечності.

- Валідація та фільтрація вхідних даних: Перевіряйте та фільтруйте вхідні дані, особливо ті, які можуть містити HTML-код. Використовуйте методи, такі як sanitize, для видалення небезпечних елементів та атрибутів.

CSRF (Cross-Site Request Forgery): CSRF-атака відбувається, коли зловмисник змушує автентифікованого користувача виконати небажані дії на веб-додатку без його попередньої згоди. Щоб запобігти CSRF-атакам, слід вживати наступні заходи:

- Використання захисту CSRF: Ruby on Rails має вбудований механізм захисту від CSRF-атак, який автоматично генерує та перевіряє токени аутентифікації при виконанні запитів POST, PUT, PATCH та DELETE. Включіть цей механізм, додавши protect_from_forgery у контролери вашого додатку.

- Використання токенів аутентифікації: Переконайтеся, що всі форми, що виконують дії, містять аутентифікаційний токен. Використовуйте form_authenticity_token для генерації та перевірки токенів.

- Обмеження методів дій: Обмежте використання методів POST, PUT, PATCH та DELETE тільки для дій, які вимагають змін на сервері. Для безпечних дій використовуйте GET-запити.

Загальний підхід до безпеки додатків на Ruby on Rails полягає в усвідомленні можливих загроз та використанні належних механізмів і заходів безпеки, що надаються фреймворком. Пам'ятайте, що безпека є постійним процесом, і важливо слідкувати за оновленнями безпеки та рекомендаціями спільноти Ruby on Rails для підтримки безпеки вашого додатку.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Вимоги ергономіки до організації робочого місця оператора ПК

Ергономіка в сучасному світі стає все важливішою в галузі інформаційних технологій, особливо при організації робочого місця оператора ПК. Завдяки науковим дослідженням та практичному досвіду було встановлено, що належне виконання ергономічних принципів позитивно впливає на здоров'я, комфорт та продуктивність працівників.

Вимоги ергономіки до організації робочого місця оператора ПК включають розташування обладнання, меблів та розміщення працівника в просторі. Основні аспекти ергономічної організації включають правильну позицію тіла під час роботи, забезпечення оптимального освітлення, вентиляції та температурного режиму, а також використання ергономічного обладнання, яке сприяє правильному положенню кистей, рук та тіла працівника.

Забезпечення комфорту та продуктивності оператора ПК є ключовими моментами, які допомагають уникнути втоми, напруги та травм, пов'язаних з роботою. Ергономічно організоване робоче місце дозволяє працівнику тривалий час займатися роботою без зайвих перерв, підвищуючи його продуктивність та якість виконуваних завдань.

Зважаючи на широке застосування комп'ютерів та інформаційних технологій у сучасному суспільстві, дотримання вимог ергономіки стає особливо важливим. Це допомагає забезпечити здоров'я та благополуччя працівників, знизити втрати від хвороб, пов'язаних з роботою, та підвищити ефективність роботи організації в цілому.

Вивчення та дотримання вимог ергономіки до організації робочого місця оператора ПК є важливим кроком у покращенні робочих умов, забезпеченні комфорту працівників та підвищенні продуктивності праці.

Загальні ергономічні вимоги для організації робочого місця користувача ПК (ДСТУ 7299:2013, ДСТУ 7951:2015). Ці вимоги встановлюють основні параметри робочого місця, оснащеного дисплеєм, і враховують особливість виконуваних робіт.

Параметри робочого місця.

Площа кабінету, в якому буде проходити робота повинна бути не менш 6м², а об'єм не менш 20м³. Для внутрішньої обробки приміщення повинні використовуватися дифузно-відбивні матеріали з коефіцієнтами відбиття для стелі – 0,7-0,8; для стін – 0,5-0,6; для підлоги – 0,3-0,5 [1].

Конструкція робочого столу повинна забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання. Конструкція крісла повинна забезпечувати підтримку раціональної робочої пози під час роботи з відео-дисплейним терміналом і ПК, дозволяти змінювати позу з метою зниження статичного напруження м'язів шийно-плечової області і спини для попередження розвитку втоми працюючого. Поверхня сидіння, спинки та інших елементів стільця (крісла) повинна бути напівм'якою, з покриттям, що не електризується, неслизьке та повітронепроникне, що забезпечує легке очищення від забруднення.

Висота робочої поверхні столу, за відсутності можливості її регулювання повинна складати 725 мм. Робочий стіл повинен мати простір для ніг висотою не менше 600 мм, шириною – не менше 500 мм, не менше 450 мм в глибину на рівні колін і на рівні простягнутої ноги – не менше 650 мм. Робоче місце має бути обладнане підставкою для ніг, має ширину не менше 300 мм, глибину не менше 400 мм, регулювання по висоті в межах 150 мм за кутом нахилу опорної поверхні підставки до 20°.

Відстань від очей користувача до екрану дисплея має становити 500-700 мм. Кут зору 10-20°, але не більше 40°; кут між верхнім краєм дисплея і рівнем очей користувача має становити не менше 10°. Кращим є розташування екрану перпендикулярно до лінії зору користувача [2].

Робочі місця по відношенню до світлових прорізів повинні розташовуватися не ближче 3 м так, щоб природне світло падало збоку, переважно зліва. Освітленість також впливає на стан здоров'я і працездатність людини. У відповідності з ДБН В.2.5-28:2018 встановлені наступні вимоги до освітленості:

Для штучного освітлення:

- комбіноване освітлення – освітленість 1500 лк;
- загальне освітлення – освітленість 400 лк.

Для природного освітлення:

- верхнє або комбіноване освітлення – коефіцієнт природної освітленості (КПО) 10%;
- бічне освітлення – КПО 3.5%.

Для суміщеного освітлення:

- верхнє або комбіноване освітлення – КПО 3-6%;
- бічне освітлення – КПО 1.1-2%.

До основних показників, що визначають умови здорової роботи, належать: фон, контраст об'єкта з фоном, видимість, показник осліпленості, коефіцієнт пульсації освітленості.

Фон характеризується коефіцієнтом відбиття. Контраст об'єкта з фоном (К) характеризується співвідношенням яскравості розглянутого об'єкта (точки, лінії, знаки) і фону. Оскільки роботи користувача ПК відносяться до категорії 1а – легкі фізичні роботи (роботи проводяться сидячи і супроводжуються незначним фізичним напруженням, з енерговитратами до 120 ккал / годину), необхідно дотримуватися наступних норм: коефіцієнт відображення більше 0,4, тобто світлий фон; контраст об'єкта з фоном великий і середній при К більше 0,2.

У полі зору користувача ПК має бути забезпечений відповідний розподіл яскравості. Відношення яскравості екрана до яскравості оточуючих його поверхонь не повинно перевищувати у робочій зоні 3:1. У зв'язку з цим дисплей ПК повинен відповідати наступним вимогам:

- яскравість свічення екрану не менше 100 кд/м;

- мінімальний розмір світної точки для кольорового дисплея не більше 0,6 мм ;

- контрастність зображення знаку – не менше 0,8;

- низькочастотне тремтіння зображення в діапазоні 0,05-1,0 Гц повинно знаходитися в межах 0,1 мм;

- екран повинен мати покриття антивідблиску;

- відеомонітор повинен бути обладнаний поворотним майданчиком, що дозволяє переміщати відеотермінал в горизонтальній і вертикальній площинах в межах 130-220 мм і змінювати кут нахилу на 10-15 мм.

Коефіцієнт відбиття світла матеріалами і обладнанням всередині приміщень має велике значення для освітлення: чим більше світла відбивається від поверхонь, тим вище освітленість. Коефіцієнт відображення відповідно повинен бути для: стелі 60-70%, стін 40-50%, підлоги 30%, для інших поверхонь 30-40% [3].

Результати досліджень показують, що найбільшою мірою негативний фізіологічний вплив на операторів ПК пов'язаний з дискомфортними зоровими умовами через неправильно спроектоване освітлення. Згідно ДБН В.2.5-28:2018 освітленість на горизонтальній площині робочого місця оператора ЕОМ повинна складати 400 лк при висоті цій площині 0,8 м над підлогою.

Вимоги до освітленості і повітряного середовища в робочій зоні.

Світловий клімат визначає зоровий дискомфорт. Запобігти шкідливому впливу освітлення можна шляхом правильного підбору системи освітлення, джерел світла (за їх спектрального складу випромінювання), світильників.

Коли штучне світло змішується з природним, рекомендується використовувати лампи за спектральним складом найбільш близькі до сонячного світла. Світильники слід вибирати з розсіювачами, а блискучі деталі освітлювального обладнання, що можуть потрапити в поле зору оператора, повинні бути замінені на матові. Розташовувати робоче місце, обладнане дисплеєм, необхідно таким чином, щоб у полі зору оператора не потрапляли вікна або освітлювальні прилади; вони не повинні знаходиться і безпосередньо за спиною оператора. Вікна в приміщеннях з дисплеями обладнають шторами з коефіцієнтом відображення 0,5 ... 0,7, стіни фарбують матовою фарбою з коефіцієнтом відображення 0,4 ... 0,6. Світловий клімат може бути поліпшений шляхом встановлення спеціальних анти-відблискових контрастних фільтрів, однак при виборі типу фільтра необхідно враховувати умови роботи з комп'ютером, оскільки оптимальні значення коефіцієнтів пропускання і дзеркального відображення фільтрів залежать від освітленості робочого місця і типу джерела світла.

Враховуючи великий вплив освітлення на працездатність оператора при роботі з комп'ютером, проведемо розрахунок необхідної освітленості в приміщенні з дисплеями при наступних умовах: гігієнічна норма освітленості на горизонтальній поверхні на рівні робочого місця оператора – 400 лк; ширина приміщення – 7 м, довжина – 8 м, висота – 3 м. Коефіцієнт відбиття від стелі – 70, від стін – 50, від робочих поверхонь – 30. Повітряне середовище – нормальне (вміст пилу, диму й кіптяви не більше 5 мг/м³).

Повітряне середовище в робочій зоні визначається мікрокліматом виробничого приміщення. Величини температури, відносної вологості та швидкості руху повітря на робочих місцях з дисплеями повинні відповідати допустимим значенням, які затверджені в наказі № 1596 від 14.07.2020 “Про

затвердження гігієнічних регламентів допустимого вмісту хімічних і біологічних речовин у повітрі робочої зони" [4] для категорії робіт 1а (легкі фізичні роботи, вироблені сидячи і супроводжуються незначною фізичною напругою до 120 ккал/год.).

Згідно з цим документом допустимі значення температури повітря в приміщенні становлять 19-25 °С, відносної вологості повітря – 55%, швидкості руху повітря на рівні особи – 0,1 м/С. При наявності досить комфортного робочого середовища атмосферний тиск може змінюватися від 84 до 107 кПа (630 ... 800 мм рт. ст.).

Шум несприятливий для людини, особливо при тривалому впливі. В оператора це виражається в зниженні працездатності (наприклад, швидкість обробки тексту зменшується на 10-15%), у прискоренні розвитку зорового стомлення, зміну відчуття кольору, підвищенні витрати енергії (на 17%). Тривалий та інтенсивний шум значно знижує продуктивність праці і призводить до зростання кількості помилок у роботі. У відділі головного економіста шум може створюватися телефонними дзвінками та розмовами, системними блоками та клавіатурою ПК. Так само джерелами шуму можуть бути системи кондиціювання та вентилювання повітря, існують і зовнішні джерела шуму (наприклад, працюють агрегати на вулиці).

Допустимі рівні звуку на робочих місцях.

Допустимі рівні звуку та еквівалентні рівні звуку на робочих місцях повинні відповідати вимогам ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку» [5]. Згідно з цими нормами в приміщенні, де працює користувач ПК для забезпечення оптимальної робочої середовища рівень шуму не повинен перевищувати 60 дБА. Основними заходами боротьби з шумом є ліквідація або ослаблення джерела шуму шляхом застосування звукопоглинаючих матеріалів у конструкціях механізмів, використання коштів звукопоглинання і раціональна планування виробничого приміщення.

Випромінювання ПК можуть бути небезпечними для здоров'я. Низькочастотні поля при тривалому опроміненні сидять біля ПК людей можуть привести до порушень самих різних фізіологічних процесів.

Потужність дози рентгенівського випромінювання в будь-якій точці простору на відстані 5 см від екрану відеомонітора при 41 годинному робочому тижні не повинна перевищувати 100 мкР/год (0,03 мкР/с), а інтенсивність ультрафіолетового випромінювання – 10 Вт/м².

В даний час випускаються вибухобезпечні відеомонітори. За способом захисту людини від ураження електричним струмом дисплеї виготовляються відповідно з 1-м класом захисту, тому кабель живлення дисплея має вилку з трьома виводами, один з яких заземлюючий.

Для забезпечення гранично допустимого рівня чинників робочого середовища на робочих місцях у необхідних випадках використовуються спеціальні засоби захисту працюючих. Способи захисту бувають активними і пасивними. Способи активного захисту засновані на виявленні джерел несприятливих факторів і вплив на них. У випадках неможливості здійснення активного захисту застосовується пасивна, за якої джерела несприятливих факторів залишаються, але здійснюються заходи, спрямовані на попередження вплив цих факторів на людину. Пасивна захист може бути колективного та індивідуального. Розглянемо колективні засоби захисту оператора ПК.

Висока температура повітря негативно позначається на функціональному стані людини. Всі основні електронні блоки ПК мають вбудовані вентилятори для забезпечення стабільних температурних режимів їх функціонування, тому при створенні комфортних умов роботи особливу увагу необхідно приділити шляхам відводу повітря (припливно-витяжної вентиляції).

Для захисту від електростатичного потенціалу і, певною мірою, від електричної складової змінного електромагнітного поля можуть бути використані згадані вище антиблискові контрастуючі фільтри на екрани дисплеїв.

4.2 Дії оператора ПК під час повітряної тривоги в ситуації воєнного стану

гідно указу Президента від 24.02.2022 в Україні введено воєнний стан. Нехтувати безпекою під час тривоги є дуже небезпечним діями, тому оператор ПК має мати алгоритм дій під час повітряної тривоги.

Повітряна тривога оголошується у разі загрози з повітря. Це може бути пов'язано з ворожими літаками, ракетами або дронами. Під час повітряної тривоги важливо дотримуватися інструкцій та негайно прямувати до укриття.

Ще до того як пролунала сирена повітряної тривоги, оператор ПК має знати де знаходися найближче укриття, та мати підготовану “екстрену валізу”.

Екстренна валіза або тривожна валіза - це сумка або рюкзак з найнеобхіднішими речами в будь-якій надзвичайній ситуації. До неї входять: копії документів, одяг, ліки, інструменти, засоби гігієни та їжа. Таку валізу можна укомплектувати заздалегідь, щоб швидко евакуюватися в небезпечній ситуації.

Оператор ПК, як і будь-який інший громадянин, повинен дотримуватися інструкцій щодо дій під час повітряної тривоги. Найголовніше правило звучить так: почули сирену – одразу прямуйте до укриття. Якщо не встигли добігти до укриття, негайно перейдіть у приміщення без вікон та скористайтеся правилом «двох стін».

Правило «двох стін» означає, що від небезпеки вас має відділяти щонайменше 2 стіни, бо одна, ймовірно, зруйнується від удару, а друга візьме на себе уламки стіни, віконного скла тощо. Тому ховатися безпечно у місці, яке розташоване за другою від фасаду опорною стіною.

Якщо немає можливості перейти в укриття, необхідно знаходитися у найбільш безпечному місці у будинку с подалі від вікон, бажано за двома стінами. Наприклад, це може бути тамбур, коридор або передпокій. Також

слід вимкнути світло, газ та воду. Якщо в будинку є газовий балон – від’єднати його, спустити тиск і покласти в безпечне місце.

Якщо у будинку стався вибух, якнайшвидше залиште будівлю самостійно. Рекмендується залишити двері відчиниими щоб рятувальники при обовязковій перевірці квартир не зрізали їх бензорізом [6].

ВИСНОВКИ

В даному академічному дослідженні була розглянута тема розробки та захисту веб-додатку для створення онлайн системи питань та відповідей (Q&A). Результати дослідження показали, що такий веб-додаток відіграє важливу роль у сучасному інформаційному суспільстві, надаючи користувачам можливість швидко знаходити відповіді на свої запитання та обмінюватися знаннями та досвідом.

Розробка веб-додатку для системи Q&A вимагає комплексного підходу, який включає аналіз потреб користувачів, проектування інтерфейсу, розробку бази даних та забезпечення безпеки. Безпека є одним з ключових аспектів у розробці такого додатку, оскільки він містить чутливу інформацію користувачів. Тому, забезпечення конфіденційності, цілісності та доступності даних є необхідною складовою частиною процесу розробки та захисту.

Веб-додаток для системи Q&A має потенціал стати платформою для навчання, співпраці та інновацій. Він надає можливість студентам, викладачам та дослідникам обмінюватися знаннями та досвідом, співпрацювати у вирішенні проблем та створювати нові можливості для наукових досліджень. Це сприяє академічному зростанню та стимулює інновації у різних сферах знань.

Отже, розробка та захист веб-додатку для створення онлайн системи питань та відповідей є важливим етапом у розвитку сучасного інформаційного суспільства. Цей додаток надає зручність та швидкість доступу до інформації, стимулює співпрацю та навчання, сприяючи створенню спільнот та забезпечуючи обмін знаннями та ідеями. Подальші дослідження у цій галузі можуть сприяти вдосконаленню даних веб-додатків та покращенню їх функціональності з метою задоволення потреб користувачів та розвитку суспільства в цілому.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна сторінка мови Ruby : вебсайт. URL: https://www.ruby-lang.org/en/ (дата звернення: 14.06.2023) .

2. Найбільша бібліотека Ruby гемів : вебсайт. URL: https://rubygems.org (дата звернення: 14.06.2023).

3. Ruby комю’ніті блог : вебсайт. URL: https://rubyflow.com (дата звернення: 14.06.2023).

4. Колекція Ruby гемів : вебсайт. URL: https://ruby.libhunt.com (дата звернення: 14.06.2023).

5. Шпаргалка по Ruby : вебсайт. URL: http://rubykoans.com (дата звернення: 14.06.2023).

6. Документація для Rails : вебсайт. URL: http://rusrails.ru (дата звернення: 14.06.2023).

7. Документація для Rails : вебсайт. URL: https://gorails.com (дата звернення: 14.06.2023).

8. Документація по середовищу програмування : вебсайт. URL: https://atom.io (дата звернення: 14.06.2023).

9. Ruby комю’ніті : вебсайт. URL: https://pivorak.com

10. Списки Тодо листів : вебсайт. URL: https://zapier.com/blog/best-todo-list-apps/ (дата звернення: 14.06.2023).

11. Стаття Rails по системі RBAC : вебсайт. URL: https://hibbard.eu/authentication-with-devise-and-cancancan-in-rails/ (дата звернення: 14.06.2023).

12. Документація для мов програмування : вебсайт. URL: https://www.linux.org.ru/ (дата звернення: 14.06.2023).

13. Stackoverflow : вебсайт. URL: https://stackoverflow.com/ (дата звернення: 14.06.2023).

14. ДСТУ 7951:2015. Дизайн і ергономіка. Крісло оператора. Загальні ергономічні вимоги.

15. ДСТУ 7299:2013. Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки.

16. ДБН В.2.5-28:2018. Природне і штучне освітлення.

17. Наказ № 1596 від 14.07.2020 Про затвердження гігієнічних регламентів допустимого вмісту хімічних і біологічних речовин у повітрі робочої зони.

18. ДСН 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку

19. ДСНС. Дії населення в умовах надзвичайних ситуацій воєнного характеру".

20. Закон України "Про захист інформації в інформаційно-телекомунікаційних системах" від 5 липня 1994 року № 80/94-ВР;

21. Закон України "Про захист персональних даних" від 1 червня 2010 року № 34;

22. НД ТЗІ 2.5-010-03 Вимоги до захисту інформації WEB-сторінки від несанкціонованого доступу;

ДОДАТОК А

Лістинг програми

Лістинг файлу app\controllers\answers_controller.rb

```
# frozen_string_literal: true

class AnswersController < ApplicationController
  include QuestionsAnswers
  include ActionView::RecordIdentifier

  before_action :set_question!
  before_action :set_answer!, except: :create
  before_action :authorize_answer!
  after_action :verify_authorized

  def update
    if @answer.update answer_update_params
      flash[:success] = t '.success'
      redirect_to question_path(@question, anchor: dom_id(@answer))
    else
      render :edit
    end
  end

  def edit; end

  def create
    @answer = @question.answers.build answer_create_params

    if @answer.save
      flash[:success] = t '.success'
      redirect_to question_path(@question)
    else
      load_question_answers(do_render: true)
    end
  end

  def destroy
    @answer.destroy
    flash[:success] = t '.success'
    redirect_to question_path(@question)
  end

  private

  def answer_create_params
    params.require(:answer).permit(:body).merge(user: current_user)
  end

  def answer_update_params
    params.require(:answer).permit(:body)
```

```
  end

  def set_question!
    @question = Question.find params[:question_id]
  end

  def set_answer!
    @answer = @question.answers.find params[:id]
  end

  def authorize_answer!
    authorize(@answer || Answer)
  end
end
```

Лістинг файлу app\controllers\application_controller.rb

```
# frozen_string_literal: true

class ApplicationController < ActionController::Base
  include Authorization
  include Pagy::Backend
  include ErrorHandling
  include Authentication
  include Internationalization
end
```

Лістинг файлу app\controllers\comments_controller.rb

```
 # frozen_string_literal: true

class CommentsController < ApplicationController
  include QuestionsAnswers
  before_action :set_commentable!
  before_action :set_question
  after_action :verify_authorized

  def create
    @comment = @commentable.comments.build comment_params
    authorize @comment

    if @comment.save
      flash[:success] = t '.success'
      redirect_to question_path(@question)
    else
      @comment = @comment.decorate
      load_question_answers do_render: true
    end
  end

  def destroy
    comment = @commentable.comments.find params[:id]
    authorize comment
```

```
    comment.destroy
    flash[:success] = t '.success'
    redirect_to question_path(@question)
  end

  private

  def comment_params
                  params.require(:comment).permit(:body).merge(user:
current_user)
  end

  def set_commentable!
          klass    =    [Question,    Answer].detect    {    |c|
params["#{c.name.underscore}_id"] }
    raise ActiveRecord::RecordNotFound if klass.blank?

                                        @commentable                =
klass.find(params["#{klass.name.underscore}_id"])
  end

  def set_question
      @question  =  @commentable.is_a?(Question)  ?  @commentable  :
@commentable.question
  end
end
```

Лістинг файлу app\controllers\pages_controller.rb

```
  # frozen_string_literal: true

class PagesController < ApplicationController
  def index; end
end
```

Лістинг файлу app\controllers\password_resets_controller.rb

```
  # frozen_string_literal: true

class PasswordResetsController < ApplicationController
  before_action :require_no_authentication
  before_action :check_user_params, only: %i[edit update]
  before_action :set_user, only: %i[edit update]

  def create
    @user = User.find_by email: params[:email]

    if @user.present?
      @user.set_password_reset_token

                                       PasswordResetMailer.with(user:
@user).reset_email.deliver_later
    end
```

```
    flash[:success] = t '.success'
    redirect_to new_session_path
  end

  def edit; end

  def update
    if @user.update user_params
      flash[:success] = t '.success'
      redirect_to new_session_path
    else
      render :edit
    end
  end

  private

  def user_params
                              params.require(:user).permit(:password,
:password_confirmation).merge(admin_edit: true)
  end

  def check_user_params
     redirect_to(new_session_path, flash: { warning: t('.fail') })
if params[:user].blank?
  end

  def set_user
    @user = User.find_by email: params[:user][:email],
                                                password_reset_token:
params[:user][:password_reset_token]

     redirect_to(new_session_path, flash: { warning: t('.fail') })
unless @user&.password_reset_period_valid?
  end
end
```

Лістинг файлу app\controllers\questions_controller.rb

```
  # frozen_string_literal: true

class QuestionsController < ApplicationController
  include QuestionsAnswers
  before_action :require_authentication, except: %i[show index]
  before_action :set_question!, only: %i[show destroy edit update]
  before_action :authorize_question!
  after_action :verify_authorized

  def show
    load_question_answers
  end

  def destroy
    @question.destroy
```

```
    flash[:success] = t('.success')
    redirect_to questions_path
  end

  def edit; end

  def update
    if @question.update question_params
      flash[:success] = t('.success')
      redirect_to questions_path
    else
      render :edit
    end
  end

  def index
    @tags = Tag.where(id: params[:tag_ids]) if params[:tag_ids]
    @pagy, @questions = pagy Question.all_by_tags(@tags)
    @questions = @questions.decorate
  end

  def new
    @question = Question.new
  end

  def create
    @question = current_user.questions.build question_params
    if @question.save
      flash[:success] = t('.success')
      redirect_to questions_path
    else
      render :new
    end
  end

  private

  def question_params
    params.require(:question).permit(:title, :body, tag_ids: [])
  end

  def set_question!
    @question = Question.find params[:id]
  end

  def authorize_question!
    authorize(@question || Question)
  end
end
```

Лістинг файлу app\controllers\sessions_controller.rb

```
  # frozen_string_literal: true
```

```
class SessionsController < ApplicationController
  before_action :require_no_authentication, only: %i[new create]
  before_action :require_authentication, only: :destroy
  before_action :set_user, only: :create

  def new; end

  def create
    if @user&.authenticate(params[:password])
      do_sign_in @user
                  flash[:success]     =     t('.success',     name:
current_user.name_or_email)
      redirect_to root_path
    else
      flash.now[:warning] = t '.invalid_creds'
      render :new
    end
  end

  def destroy
    sign_out
    flash[:success] = t '.success'
    redirect_to root_path
  end

  private

  def set_user
    @user = User.find_by email: params[:email]
  end

  def do_sign_in(user)
    sign_in user
    remember(user) if params[:remember_me] == '1'
  end
end
```

Лістинг файлу app\controllers\users_controller.rb

```
  # frozen_string_literal: true

class UsersController < ApplicationController
  before_action :require_no_authentication, only: %i[new create]
  before_action :require_authentication, only: %i[edit update]
  before_action :set_user!, only: %i[edit update]
  before_action :authorize_user!
  after_action :verify_authorized

  def edit; end

  def update
    if @user.update user_params
      flash[:success] = t '.success'
```

```
      redirect_to edit_user_path(@user)
    else
      render :edit
    end
  end

  def new
    @user = User.new
  end

  def create
    @user = User.new user_params
    if @user.save
      sign_in @user
                  flash[:success]     =     t('.success',     name:
current_user.name_or_email)
      redirect_to root_path
    else
      render :new
    end
  end

  private

  def set_user!
    @user = User.find params[:id]
  end

  def user_params
        params.require(:user).permit(:email,    :name,    :password,
:password_confirmation,
                                  :old_password)
  end

  def authorize_user!
    authorize(@user || User)
  end
end
```

Лістинг файлу app\controllers\admin\base_controller.rb

```
  # frozen_string_literal: true

module Admin
  class BaseController < ApplicationController
    def authorize(record, query = nil)
      super([:admin, record], query)
    end
  end
end
```

Лістинг файлу app\controllers\admin\users_controller.rb

```
  # frozen_string_literal: true

module Admin
```

```ruby
  class UsersController < BaseController
    before_action :require_authentication
    before_action :set_user!, only: %i[edit update destroy]
    before_action :authorize_user!
    after_action :verify_authorized

    def index
      respond_to do |format|
        format.html do
          @pagy, @users = pagy User.order(created_at: :desc)
        end

        format.zip do
          UserBulkExportJob.perform_later current_user
          flash[:success] = t '.success'
          redirect_to admin_users_path
        end
      end
    end

    def create
      if params[:archive].present?
        UserBulkImportJob.perform_later create_blob, current_user
        flash[:success] = t '.success'
      end

      redirect_to admin_users_path
    end

    def edit; end

    def update
      if @user.update user_params
        flash[:success] = t '.success'
        redirect_to admin_users_path
      else
        render :edit
      end
    end

    def destroy
      @user.destroy
      flash[:success] = t '.success'
      redirect_to admin_users_path
    end

    private

    def create_blob
      file = File.open params[:archive]
      result = ActiveStorage::Blob.create_and_upload! io: file,
                                                      filename:
params[:archive].original_filename
      file.close
```

```
      result.key
    end

    def set_user!
      @user = User.find params[:id]
    end

    def user_params
      params.require(:user).permit(
        :email, :name, :password, :password_confirmation, :role
      ).merge(admin_edit: true)
    end

    def authorize_user!
      authorize(@user || User)
    end
  end
end
```

Лістинг файлу app\controllers\api\tags_controller.rb

```
  # frozen_string_literal: true

module Api
  class TagsController < ApplicationController
    def index
      tags = Tag.arel_table
      @tags = Tag.where(tags[:title].matches("%#{params[:term]}%"))

      render json: TagBlueprint.render(@tags)
    end
  end
end
```

Лістинг файлу app\decorators\answer_decorator.rb

```
  # frozen_string_literal: true

class AnswerDecorator < ApplicationDecorator
  delegate_all
  decorates_association :user

  def formatted_created_at
    l created_at, format: :long
  end
end
```

Лістинг файлу app\decorators\application_decorator.rb

```
  # frozen_string_literal: true

class ApplicationDecorator < Draper::Decorator
  # Define methods for all decorated objects.
  # Helpers are accessed through `helpers` (aka `h`). For example:
```

```
  #
  #   def percent_amount
  #     h.number_to_percentage object.amount, precision: 2
  #   end
end
```

Лістинг файлу app\decorators\comment_decorator.rb

```
  # frozen_string_literal: true

class CommentDecorator < ApplicationDecorator
  delegate_all
  decorates_association :user

  def for?(commentable)
    commentable = commentable.object if commentable.decorated?

    commentable == self.commentable
  end
end
```

Лістинг файлу app\decorators\question_decorator.rb

```
  # frozen_string_literal: true

class QuestionDecorator < ApplicationDecorator
  delegate_all
  decorates_association :user

  def formatted_created_at
    l created_at, format: :long
  end
end
```

Лістинг файлу app\decorators\user_decorator.rb

```
   # frozen_string_literal: true

class UserDecorator < ApplicationDecorator
  delegate_all

  def name_or_email
    return name if name.present?

    email.split('@')[0]
  end

  def gravatar(size: 30, css_class: '')
                                                    h.image_tag
"https://www.gravatar.com/avatar/#{gravatar_hash}.jpg?s=#{size}",
                class: "rounded #{css_class}", alt: name_or_email
  end
end
```

Лістинг файлу app\helpers\admin\users_helper.rb

```
  # frozen_string_literal: true

module Admin
  module UsersHelper
    def user_roles
      User.roles.keys.map do |role|
        [t(role, scope: 'global.user.roles'), role]
      end
    end
  end
end
```

Лістинг файлу app\helpers\application_helper.rb

```
   # frozen_string_literal: true

module ApplicationHelper
  include Pagy::Frontend

  def pagination(obj)
    # rubocop:disable Rails/OutputSafety
    raw(pagy_bootstrap_nav(obj)) if obj.pages > 1
    # rubocop:enable Rails/OutputSafety
  end

  def nav_tab(title, url, options = {})
    current_page = options.delete :current_page

     css_class = current_page == title ? 'text-secondary' : 'text-
white'

    options[:class] = if options[:class]
                        "#{options[:class]} #{css_class}"
                      else
                        css_class
                      end

    link_to title, url, options
  end

  def currently_at(current_page = '')
       render  partial:  'shared/menu',  locals:  {  current_page:
current_page }
  end

  def full_title(page_title = '')
    base_title = 'AskIt'
    if page_title.present?
      "#{page_title} | #{base_title}"
    else
      base_title
    end
  end
```

```
end
```

Лістинг файлу app\jobs\user_bulk_export_job.rb

```
   # frozen_string_literal: true

class UserBulkExportJob < ApplicationJob
  queue_as :default

  def perform(initiator)
    stream = UserBulkExportService.call

    Admin::UserMailer.with(user: initiator, stream: stream)
                     .bulk_export_done.deliver_now
  end
end
```

Лістинг файлу app\jobs\user_bulk_import_job.rb

```
   # frozen_string_literal: true

class UserBulkImportJob < ApplicationJob
  queue_as :default

  def perform(archive_key, initiator)
    UserBulkImportService.call archive_key
  rescue StandardError => e
            Admin::UserMailer.with(user:     initiator,     error:
e).bulk_import_fail.deliver_now
  else
                                          Admin::UserMailer.with(user:
initiator).bulk_import_done.deliver_now
  end
end
```

Лістинг файлу app\mailers\application_mailer.rb

```
   # frozen_string_literal: true

class ApplicationMailer < ActionMailer::Base
  default from: 'admin@askit.com'
  layout 'mailer'
end
```

Лістинг файлу app\mailers\password_reset_mailer.rb

```
   # frozen_string_literal: true

class PasswordResetMailer < ApplicationMailer
  def reset_email
    @user = params[:user]
```

```
                mail        to:        @user.email,        subject:
I18n.t('password_reset_mailer.reset_email.subject')
  end
end
```

Лістинг файлу app\models\answer.rb

```
   # frozen_string_literal: true

class Answer < ApplicationRecord
  include Authorship
  include Commentable

  belongs_to :question
  belongs_to :user

  validates :body, presence: true, length: { minimum: 5 }
end
```

Лістинг файлу app\models\application_record.rb

```
   # frozen_string_literal: true

class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Лістинг файлу app\models\comment.rb

```
   # frozen_string_literal: true

class Comment < ApplicationRecord
  include Authorship

  belongs_to :commentable, polymorphic: true
  belongs_to :user

  validates :body, presence: true, length: { minimum: 2 }
end
```

Лістинг файлу app\models\question_tag.rb

```
   # frozen_string_literal: true

class QuestionTag < ApplicationRecord
  belongs_to :question
  belongs_to :tag
end
```

Лістинг файлу app\models\question.rb

```
   # frozen_string_literal: true
```

```
class Question < ApplicationRecord
  include Authorship
  include Commentable

  has_many :answers, dependent: :destroy
  belongs_to :user
  has_many :question_tags, dependent: :destroy
  has_many :tags, through: :question_tags

  validates :title, presence: true, length: { minimum: 2 }
  validates :body, presence: true, length: { minimum: 2 }

  scope :all_by_tags, lambda { |tags|
    questions = includes(:user)
    questions = if tags
                                    questions.joins(:tags).where(tags:
tags).preload(:tags)
                else
                  questions.includes(:question_tags, :tags)
                end

    questions.order(created_at: :desc)
  }
end
```

Лістинг файлу app\models\tag.rb

```
   # frozen_string_literal: true

class Tag < ApplicationRecord
  has_many :question_tags, dependent: :destroy
  has_many :questions, through: :question_tags

  validates :title, presence: true, uniqueness: true
end
```

Лістинг файлу app\models\user.rb

```
   # frozen_string_literal: true

class User < ApplicationRecord
  include Recoverable
  include Rememberable

  enum role: { basic: 0, moderator: 1, admin: 2 }, _suffix: :role

  attr_accessor :old_password, :admin_edit

  has_secure_password validations: false
```

```
  has_many :questions, dependent: :destroy
  has_many :answers, dependent: :destroy

  validate :password_presence
    validate   :correct_old_password,   on:   :update,   if:   ->   {
password.present? && !admin_edit }
  validates :password, confirmation: true, allow_blank: true,
                       length: { minimum: 8, maximum: 70 }

    validates   :email,   presence:   true,   uniqueness:   true,
'valid_email_2/email': true
  validate :password_complexity
  validates :role, presence: true

  before_save :set_gravatar_hash, if: :email_changed?

  def guest?
    false
  end

  def author?(obj)
    obj.user == self
  end

  private

  def set_gravatar_hash
    return if email.blank?

    hash = Digest::MD5.hexdigest email.strip.downcase
    self.gravatar_hash = hash
  end

  def digest(string)
    cost = if ActiveModel::SecurePassword
              .min_cost
             BCrypt::Engine::MIN_COST
           else
             BCrypt::Engine.cost
           end
    BCrypt::Password.create(string, cost: cost)
  end

  def correct_old_password
                                          return                    if
BCrypt::Password.new(password_digest_was).is_password?(old_password
)

    errors.add :old_password, 'is incorrect'
  end

  def password_complexity
```

```
                    #         Regexp         extracted         from
https://stackoverflow.com/questions/19605150/regex-for-password-
must-contain-at-least-eight-characters-at-least-one-number-a
    return if password.blank? || password =~ /(?=.*?[A-Z])(?=.*?[a-
z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-])/

    msg = 'complexity requirement not met. Length should be 8-70
characters and ' \
          'include: 1 uppercase, 1 lowercase, 1 digit and 1 special
character'
    errors.add :password, msg
  end

  def password_presence
    errors.add(:password, :blank) if password_digest.blank?
  end
end
```

Лістинг файлу app\policies\admin\user_policy.rb

```
   # frozen_string_literal: true

module Admin
  class UserPolicy < ApplicationPolicy
    def create?
      user.admin_role?
    end

    def update?
      user.admin_role?
    end

    def index?
      user.admin_role?
    end

    def show?
      user.admin_role?
    end

    def destroy?
      user.admin_role?
    end
  end
end
```

Лістинг файлу app\policies\answer_policy.rb

```
   # frozen_string_literal: true

class AnswerPolicy < ApplicationPolicy
  def create?
    !user.guest?
```

```
  end

  def update?
            user.admin_role?     ||     user.moderator_role?     ||
user.author?(record)
  end

  def index?
    true
  end

  def show?
    true
  end

  def destroy?
    user.admin_role? || user.author?(record)
  end
end
```

Лістинг файлу app\policies\application_policy.rb

```
   # frozen_string_literal: true

class ApplicationPolicy
  attr_reader :user, :record

  def initialize(user, record)
    @user = user || GuestUser.new
    @record = record
  end

  def index?
    false
  end

  def show?
    false
  end

  def create?
    false
  end

  def new?
    create?
  end

  def update?
    false
  end

  def edit?
```

```
    update?
  end

  def destroy?
    false
  end

  class Scope
    def initialize(user, scope)
      @user = user
      @scope = scope
    end

    def resolve
      scope.all
    end

    private

    attr_reader :user, :scope
  end
end
```

Лістинг файлу app\policies\comment_policy.rb

```
   # frozen_string_literal: true

class CommentPolicy < ApplicationPolicy
  def create?
    !user.guest?
  end

  def update?
            user.admin_role?     ||     user.moderator_role?     ||
user.author?(record)
  end

  def index?
    true
  end

  def show?
    true
  end

  def destroy?
    user.admin_role? || user.author?(record)
  end
end
```

Лістинг файлу app\policies\question_policy.rb

```
   # frozen_string_literal: true
```

```
class QuestionPolicy < ApplicationPolicy
  def create?
    !user.guest?
  end

  def update?
            user.admin_role?     ||     user.moderator_role?     ||
user.author?(record)
  end

  def destroy?
    user.admin_role? || user.author?(record)
  end

  def index?
    true
  end

  def show?
    true
  end
end
```

Лістинг файлу app\policies\user_policy.rb

```
   # frozen_string_literal: true

class UserPolicy < ApplicationPolicy
  def create?
    user.guest?
  end

  def update?
    record == user
  end

  def index?
    false
  end

  def show?
    true
  end

  def destroy?
    false
  end
end
```

Лістинг файлу app\views\admin\users_form.html.erb

```
   <%= render 'shared/errors', object: user %>
```

```
<%= form_with model: [:admin, user] do |f| %>
  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :email %>
    </div>

    <div class="col-sm-10">
       <%= f.email_field :email, class: 'form-control form-control-
lg' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :name %>
    </div>

    <div class="col-sm-10">
       <%= f.text_field :name, class: 'form-control form-control-lg'
%>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :role %>
    </div>

    <div class="col-sm-10">
      <%= f.select :role, user_roles, {}, class: 'form-select form-
select-lg' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :password %>
    </div>

    <div class="col-sm-10">
       <%= f.password_field :password, class: 'form-control form-
control-lg' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :password_confirmation %>
    </div>

    <div class="col-sm-10">
       <%= f.password_field :password_confirmation, class: 'form-
control form-control-lg' %>
```

```
    </div>
  </div>

  <% t_key = (user.new_record? ? "register" : "save") %>
   <%= f.submit t(t_key, scope: 'users.global.forms'), class: 'btn
btn-primary btn-lg' %>
<% end %>
```

Лістинг файлу app\views\admin\users_user.html.erb

```
   <tr>
  <th scope="row"><%= user.id %></th>
  <td><%= user.name %></td>
  <td><%= user.email %></td>
  <td><%= user.role %></td>
  <td><%= user.created_at %></td>
  <td>
     <%= link_to 'Edit', edit_admin_user_path(user), class: 'btn
btn-secondary' %>
     <%= link_to 'Delete', admin_user_path(user), class: 'btn btn-
danger',
      data: {method: :delete, confirm: t('global.dialog.you_sure')}
%>
  </td>
</tr>
```

Лістинг файлу app\views\admin\users\edit.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.users') %>

<h1 class="mb-4"><%= t('.title') %></h1>

<%= render 'form', user: @user %>
```

Лістинг файлу app\views\admin\users\index.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.users') %>

<h1 class="mb-4"><%= t '.title' %></h1>

<div class="btn-group mb-3">
    <%=   link_to   t('.download_zipped'),   admin_users_path(format:
:zip),
           class:    'btn    btn-secondary',    data:    {confirm:
t('global.dialog.you_sure')} %>
</div>

<%= pagination @pagy %>

<% scope = 'activerecord.attributes.user' %>
<table class="table table-striped table-hover">
  <thead>
    <tr>
      <th scope="col">#</th>
```

```
      <th scope="col"><%= t 'name', scope: scope %></th>
      <th scope="col"><%= t 'email', scope: scope %></th>
      <th scope="col"><%= t 'role', scope: scope %></th>
      <th scope="col"><%= t 'created_at', scope: scope %></th>
      <th scope="col"></th>
    </tr>
  </thead>
  <tbody>
    <%= render @users %>
  </tbody>
</table>

<%= pagination @pagy %>

<h2 class="mb-3"><%= t '.upload_zipped' %></h2>

<%= form_with url: admin_users_path do |f| %>
  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :archive, t('.upload_form.archive') %>
    </div>

    <div class="col-sm-10">
      <%= f.file_field :archive, class: 'form-control' %>
    </div>
  </div>

  <%= f.submit t('.upload_form.submit'), class: 'btn btn-primary'
%>
<% end %>
```

Лістинг файлу app\views\answers_answer.html.erb

```
    <%= tag.article class: 'my-3 card border-0 border-top', id:
dom_id(answer) do %>
  <div class="row g-0">
    <div class="col-sm-auto text-sm-center align-self-center">
      <%= answer.user.gravatar size: 50, css_class: 'd-block' %>
      <%= answer.user.name_or_email %>
    </div>
    <div class="col-sm">
      <div class="card-body">
        <section class="card-text mb-3">
          <div class="col-sm-9">
             <small><time datetime="<%= answer.formatted_created_at
%>">
              <%= answer.formatted_created_at %>
            </time></small>

            <div class="mt-2">
              <%= sanitize answer.body %>
            </div>
          </div>
        </section>
```

```
        <% if policy(answer).edit? %>
                          <%=    link_to    t('global.button.edit'),
edit_question_answer_path(question, answer),
             class: 'btn btn-info btn-sm' %>
        <% end %>

        <% if policy(answer).destroy? %>
                         <%=    link_to    t('global.button.delete'),
question_answer_path(question, answer), class: 'btn btn-danger btn-
sm',
                              data:    {method:    :delete,    confirm:
t('global.dialog.you_sure')} %>
        <% end %>
      </div>
    </div>
  </div>
<% end %>

<%= render 'comments/commentable', commentable: answer, comment:
@comment,
  html_id: dom_id(answer, 'comment_form') %>
```

Лістинг файлу app\views\answers_form.html.erb

```
   <%= render 'shared/errors', object: @answer %>

<%= form_with model: [@question, @answer] do |f| %>
  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :body %>
    </div>

    <div class="col-sm-10">
       <%= f.text_area :body, class: 'form-control', required: true
%>
    </div>
  </div>

  <%= f.submit t('global.button.submit'), class: 'btn btn-primary'
%>
<% end %>
```

Лістинг файлу app\views\answers\edit.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.questions') %>

<h1><%= t '.title' %></h1>

<%= render 'form' %>
```

Лістинг файлу app\views\comments_comment.html.erb

```
   <%= tag.article class: 'mb-3 card border-0', id: dom_id(comment)
do %>
  <div class="row g-0">
    <div class="col-sm-auto text-sm-center align-self-center">
      <%= comment.user.gravatar size: 30, css_class: 'd-block' %>
      <%= comment.user.name_or_email %>
    </div>
    <div class="col-sm">
      <div class="card-body">
        <section class="card-text mb-3">
          <div class="col-sm-9">
            <p>
              <%= sanitize comment.body %>
            </p>

            <% if policy(comment).destroy? %>
              <%= link_to t('global.button.delete'),
                polymorphic_path([comment.commentable, comment]),
                class: 'btn btn-danger btn-sm',
                                data:   {method:   :delete,   confirm:
t('global.dialog.you_sure')} %>
            <% end %>
          </div>
        </section>
      </div>
    </div>
  </div>
<% end %>
```

Лістинг файлу app\views\comments_commentable.html.erb

```
   <% is_current_comment = comment&.for?(commentable) %>
<%    comment_builder    =    is_current_comment    ?    comment    :
commentable.comments.build %>

<%= link_to t('questions.show.comments'), "##{html_id}",
  class: 'btn btn-primary', data: {'bs-toggle': 'collapse'} %>

<%= tag.div class: "collapse #{'show' if is_current_comment}", id:
html_id do %>
  <% if policy(comment_builder).create? %>
    <%= form_with model: [commentable, comment_builder],
    class: 'pt-3 border-top my-2' do |f| %>
      <% if is_current_comment %>
        <%= render 'shared/errors', object: comment %>
      <% end %>

      <div class="mb-3 row">
        <div class="col-sm-2 col-form-label">
          <%= f.label :body %>
        </div>

        <div class="col-sm-10">
```

```
          <%= f.text_area :body, class: 'form-control form-control-
sm' %>
        </div>
      </div>

       <%= f.submit t('global.button.submit'), class: 'btn btn-sm
btn-primary' %>
    <% end %>
  <% end %>

  <%= render commentable.comments.includes(:user).decorate %>
<% end %>
```

Лістинг файлу app\views\layouts\application.html.erb

```
   <!DOCTYPE html>
<html lang="<%= I18n.locale %>">
  <head>
    <meta charset="utf-8">
    <title><%= full_title(yield(:page_title)) %></title>
       <meta  name="viewport"  content="width=device-width,initial-
scale=1">
    <%= csrf_meta_tags %>
    <%= csp_meta_tag %>
     <%= stylesheet_pack_tag 'application', media: 'all', 'data-
turbolinks-track': 'reload' %>
    <%= javascript_pack_tag 'application', 'data-turbolinks-track':
'reload' %>
  </head>

  <body data-lang="<%= I18n.locale %>">
    <%= yield :main_menu %>

    <main class="container mt-3">
      <% flash.each do |k, v| %>
        <%= tag.div v, class: "alert alert-#{k}", role: 'alert' %>
      <% end %>

      <%= yield %>
    </main>
  </body>
</html>
```

Лістинг файлу app\views\layouts\mailer.html.erb

```
   <!DOCTYPE html>
<html>
  <head>
          <meta    http-equiv="Content-Type"    content="text/html;
charset=utf-8" />
    <style>
      /* Email styles need to be inline */
```

```
    </style>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

Лістинг файлу app\views\layouts\mailer.text.erb

```
   <%= yield %>
```

Лістинг файлу app\views\pages\index.html.erb

```
   <% currently_at t('menu.home') %>
<% provide :page_title, t('.title') %>

<h1><%= t '.title' %></h1>

&copy; @leeroy <%= Time.now.year %>
```

Лістинг файлу app\views\password_reset_mailer\reset_email.en.html.erb

```
   C<p>Someone has requested password reset for your account.</p>

<p>
  <%= link_to 'Reset my password',
              edit_password_reset_url(user:    {password_reset_token:
@user.password_reset_token,
                              email: @user.email},
                              locale: :en),
      target: '_blank' %>
</p>
```

Лістинг файлу app\views\password_reset_mailer\reset_email.en.text.erb

```
         Someone has requested password reset for your account.
Please visit this URL to reset:

<%=  url_for  edit_password_reset_url(user:  {password_reset_token:
@user.password_reset_token,
            email: @user.email}, locale: :en) %>
```

Лістинг файлу app\views\password_reset_mailer\reset_email.ua.html.erb

```
   <p>Хтось запросив сброс паролю.</p>

<p>
  <%= link_to 'Сбросить мой пароль',
              edit_password_reset_url(user:    {password_reset_token:
@user.password_reset_token,
                              email: @user.email},
                              locale: :ru),
      target: '_blank' %>
</p>
```

Лістинг файлу app\views\password_reset_mailer\reset_email.ua.text.erb

```
   Хтось запросив скидування паролю, перейдіть по силці:

<%=  url_for  edit_password_reset_url(user:  {password_reset_token:
@user.password_reset_token,
            email: @user.email}, locale: :ru) %>
```

Лістинг файлу app\views\password_resets\edit.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.log_in') %>

<h1 class="mb-4"><%= t('.title') %></h1>

<%= render 'shared/errors', object: @user %>
<%=  form_with  model:  @user,  url:  password_reset_path,  method:
:patch do |f| %>
      <%=     f.hidden_field     :password_reset_token,     value:
@user.password_reset_token %>
  <%= f.hidden_field :email, value: @user.email %>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :password %>
    </div>

    <div class="col-sm-10">
       <%=  f.password_field  :password,  class:  'form-control  form-
control-lg' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
                <%=    f.label    :password_confirmation,    for:
'user[password_confirmation]' %>
    </div>

    <div class="col-sm-10">
       <%=  f.password_field  :password_confirmation,  class:  'form-
control form-control-lg' %>
    </div>
  </div>

  <%=  f.submit  t('.form.submit'),  class:  'btn  btn-primary  btn-lg'
%>
<% end %>
```

Лістинг файлу app\views\password_resets\new.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.log_in') %>

<h1 class="mb-4"><%= t('.title') %></h1>
```

```
<%= form_with url: password_reset_path do |f| %>
  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :email, t('.form.email') %>
    </div>

    <div class="col-sm-10">
      <%= f.email_field :email, value: params[:email],
        class: 'form-control form-control-lg', required: true %>
    </div>
  </div>

  <%= f.submit t('.form.submit'), class: 'btn btn-primary btn-lg'
%>
  <%= link_to t('.form.back'), new_session_path, class: 'btn btn-
secondary' %>
<% end %>
```

Лістинг файлу app\views\questions_form.html.erb

```
   <%= render 'shared/errors', object: question %>

<%= form_with model: question do |f| %>
  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :title %>
    </div>

    <div class="col-sm-10">
      <%= f.text_field :title, class: 'form-control' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :body %>
    </div>

    <div class="col-sm-10">
      <%= f.text_area :body, class: 'form-control' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :tags %>
    </div>

    <div class="col-sm-10">
      <%= f.collection_select :tag_ids, question.tags, :id, :title,
{}, multiple: true,
                class:   'js-multiple-select',   data:   {'ajax-url':
'/api/tags'} %>
```

```
    </div>
  </div>

  <%= f.submit t('global.button.submit'), class: 'btn btn-primary'
%>
<% end %>
```

Лістинг файлу app\views\questions_question.html.erb

```
   <article class="card my-3">
  <section class="card-header">
    <%= question.user.gravatar %>
    <%= question.user.name_or_email %>
  </section>

  <div class="card-body">
    <h4><%= link_to question.title, question_path(question) %></h2>

    <section class="card-text">
      <%= tag.time datetime: question.formatted_created_at do %>
        <small><%= question.formatted_created_at %></small>
      <% end %>

      <div class="my-2">
        <%= render question.tags %>
      </div>

      <p class="my-2">
           <%= truncate strip_tags(question.body), length: 150,
omission: t('global.text.omission') %>
      </p>
    </section>

     <%= link_to t('global.button.show'), question_path(question),
class: 'btn btn-primary' %>

    <% if policy(question).edit? %>
                     <%=      link_to      t('global.button.edit'),
edit_question_path(question), class: 'btn btn-secondary' %>
    <% end %>

    <% if policy(question).destroy? %>
                    <%=      link_to      t('global.button.delete'),
question_path(question), class: 'btn btn-danger',
                         data:     {method:     :delete,     confirm:
t('global.dialog.you_sure')} %>
    <% end %>
  </div>
</article>
```

Лістинг файлу app\views\questions\edit.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.questions') %>
```

```
<h1 class="mb-4"><%= t '.title' %></h1>

<%= render 'form', question: @question %>
```

Лістинг файлу app\views\questions\index.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.questions') %>

<h1 class="mb-4"><%= t('.title') %></h1>

<section class="mb-5">
  <h2><%= t '.search_by_tags' %></h2>

  <%= form_with url: questions_path, method: :get do |f| %>
    <div class="mb-3">
      <%= f.collection_select :tag_ids, (@tags || []), :id, :title,
{selected: params[:tag_ids]},
        multiple: true, required: true,
               class:  'js-multiple-select',  data:  {'ajax-url':
'/api/tags'} %>
    </div>

      <%=  f.submit  t('global.button.submit'),  class:  'btn  btn-
primary' %>
  <% end %>
</section>

<% if policy(:question).new? %>
  <%= link_to t('.new'), new_question_path, class: 'btn btn-primary
btn-lg mb-3' %>
<% end %>

<%= pagination @pagy %>

<%= render @questions %>

<%= pagination @pagy %>
```

Лістинг файлу app\views\questions\new.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.questions') %>

<h1 class="mb-4"><%= t('.title') %></h1>

<%= render 'form', question: @question %>
```

Лістинг файлу app\views\questions\show.html.erb

```
   <% provide :page_title, @question.title %>
<% currently_at t('menu.questions') %>

<h1 class="mb-4"><%= @question.title %></h1>
```

```
<time datetime="<%= @question.formatted_created_at %>">
  <%= @question.formatted_created_at %>
</time>

<div class="lead my-3">
  <%= sanitize @question.body %>
</div>

<div class="mb-3">
  <%= @question.user.gravatar %>
  <%= @question.user.name_or_email %>
</div>

<%= render 'comments/commentable', commentable: @question, comment:
@comment,
  html_id: 'questionComments' %>

<div class="border-top border-bottom py-3 mt-3">
  <div class="btn-group">
    <% if policy(@question).edit? %>
                      <%=      link_to      t('global.button.edit'),
edit_question_path(@question), class: 'btn btn-secondary' %>
    <% end %>

    <% if policy(@question).destroy? %>
                     <%=      link_to      t('global.button.delete'),
question_path(@question), class: 'btn btn-danger',
                        data:     {method:     :delete,     confirm:
t('global.dialog.you_sure')} %>
    <% end %>
  </div>
</div>

<h2 class="my-3"><%= t '.write_answer' %></h2>

<%= render 'answers/form' %>

<h2 class="mt-5 mb-3"><%= t '.answers' %></h2>

<%= pagination @pagy %>

<%= render partial: 'answers/answer', collection: @answers,
  as: :answer, locals: {question: @question} %>

<%= pagination @pagy %>
```

Лістинг файлу app\views\sessions\new.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.log_in') %>

<h1 class="mb-4"><%= t('.title') %></h1>
```

```
<%= form_with url: session_path do |f| %>
  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :email, t('.form.email') %>
    </div>

    <div class="col-sm-10">
        <%= f.email_field :email, value: params[:email], class:
'form-control form-control-lg' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :password, t('.form.password') %>
    </div>

    <div class="col-sm-10">
       <%= f.password_field :password, class: 'form-control form-
control-lg' %>
    </div>
  </div>

  <div class="row mb-3">
    <div class="offset-sm-2 col-sm-10">
      <div class="form-check">
          <%= f.check_box :remember_me, class: 'form-check-input',
value: '1' %>
          <%= f.label :remember_me, t('.form.remember_me'), class:
'form-check-label' %>
      </div>
    </div>
  </div>

   <%= f.submit t('.form.submit'), class: 'btn btn-primary btn-lg'
%>
  <%= link_to t('.form.forgot_password'), new_password_reset_path,
    class: 'btn btn-secondary' %>
<% end %>
```

Лістинг файлу app\views\shared_errors.html.erb

```
   <% if object&.errors&.full_messages&.any? %>
  <div class="alert alert-danger">
    <ul class="mb-0">
      <% object.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

Лістинг файлу app\views\shared_menu.html.erb

```
   <%= provide :main_menu do %>
  <header class="p-3 bg-dark text-white">
    <div class="container">
        <nav class="d-flex flex-wrap align-items-center justify-
content-center justify-content-lg-start">
         <ul class="nav col-12 col-lg-auto me-lg-auto mb-2 justify-
content-center mb-md-0">
          <li><%= nav_tab t('menu.home'), root_path,
               class: 'nav-link px-2', current_page: current_page
%></li>
          <li><%= nav_tab t('menu.questions'), questions_path,
               class: 'nav-link px-2', current_page: current_page
%></li>

          <% if policy([:admin, :user]).index? %>
            <li><%= nav_tab t('menu.users'), admin_users_path,
                 class: 'nav-link px-2', current_page: current_page
%></li>
          <% end %>
        </ul>

        <ul class="nav col-12 col-lg-auto mb-2 mb-md-0">
          <li class="dropdown">
             <%= link_to '#', class: 'nav-link px-2 dropdown-toggle
text-white',
              data: {"bs-toggle": 'dropdown'} do %>
               <%= tag.div '', class: "flag #{I18n.locale}-flag mt-
1" %>
              <%= t I18n.locale %>
            <% end %>

            <ul class="dropdown-menu">
              <% I18n.available_locales.each do |locale| %>
                <li>
                  <% if I18n.locale == locale %>
                     <%= tag.span t(locale), class: 'dropdown-item'
%>
                  <% else %>
                    <%= link_to t(locale), url_for(locale: locale),
                      class: 'dropdown-item' %>
                  <% end %>
                </li>
              <% end %>
            </ul>
          </li>
        </ul>

        <ul class="nav col-12 col-lg-auto mb-2 mb-md-0">
          <% if user_signed_in? %>
            <li class="dropdown">
              <%= link_to '#',
                class: 'nav-link px-2 dropdown-toggle text-white',
                data: {"bs-toggle": 'dropdown'} do %>
                <%= current_user.gravatar %>
```

```
              <%= current_user.name_or_email %>
            <% end %>

            <ul class="dropdown-menu">
              <li>
                              <%=  link_to  t('menu.edit_profile'),
edit_user_path(current_user),
                  class: 'dropdown-item' %>
              </li>

              <li><hr class="dropdown-divider"></li>

              <li>
                <%= link_to t('menu.log_out'), session_path,
                   class: 'dropdown-item', data: {method: :delete}
%>
              </li>
            </ul>
          </li>
        <% else %>
          <li><%= nav_tab t('menu.sign_up'), new_user_path,
               class: 'nav-link px-2', current_page: current_page
%></li>

          <li><%= nav_tab t('menu.log_in'), new_session_path,
               class: 'nav-link px-2', current_page: current_page
%></li>
        <% end %>
      </ul>
    </nav>
  </div>
</header>
<% end %>
```

Лістинг файлу app\views\tags_tag.html.erb

```
         <%= link_to questions_path(tag_ids: tag),
  class: 'badge rounded-pill bg-light text-dark d-inline-block px-2
pt-1 pb-2 me-1' do %>
  <small><%= tag.title %></small>
<% end %>
```

Лістинг файлу app\views\users_form.html.erb

```
   <%= render 'shared/errors', object: user %>

<%= form_with model: user do |f| %>
  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :email %>
    </div>

    <div class="col-sm-10">
       <%= f.email_field :email, class: 'form-control form-control-
lg' %>
```

```
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :name %>
    </div>

    <div class="col-sm-10">
      <%= f.text_field :name, class: 'form-control form-control-lg'
%>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :role %>
    </div>

    <div class="col-sm-10">
          <%=  f.text_field  :role,  value:  t(user.role,  scope:
'global.user.roles'),
         class: 'form-control-plaintext form-control-lg',readonly:
true %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :password %>
    </div>

    <div class="col-sm-10">
       <%= f.password_field :password, class: 'form-control form-
control-lg' %>
    </div>
  </div>

  <div class="mb-3 row">
    <div class="col-sm-2 col-form-label">
      <%= f.label :password_confirmation %>
    </div>

    <div class="col-sm-10">
       <%= f.password_field :password_confirmation, class: 'form-
control form-control-lg' %>
    </div>
  </div>

  <% unless user.new_record? %>
    <div class="mb-3 row">
      <div class="col-sm-2 col-form-label">
        <%= f.label :old_password %>
      </div>
```

```
      <div class="col-sm-10">
          <%= f.password_field :old_password, class: 'form-control
form-control-lg' %>
      </div>
    </div>
  <% end %>

  <% t_key = (user.new_record? ? "register" : "save") %>
   <%= f.submit t(t_key, scope: 'users.global.forms'), class: 'btn
btn-primary btn-lg' %>
<% end %>
```

Лістинг файлу app\views\users\edit.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.edit_profile') %>

<h1 class="mb-4"><%= t('.title') %></h1>

<%= render 'form', user: @user %>
```

Лістинг файлу app\views\users\new.html.erb

```
   <% provide :page_title, t('.title') %>
<% currently_at t('menu.sign_up') %>

<h1 class="mb-4"><%= t('.title') %></h1>

<%= render 'form', user: @user %>
```

Лістинг файлу config\database.yml

```
   # SQLite. Versions 3.8.0 and up are supported.
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem 'sqlite3'
#
default: &default
  adapter: sqlite3
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: db/test.sqlite3
```

```
production:
  <<: *default
  database: db/production.sqlite3
```

Лістинг файлу Gemfile

```
   # frozen_string_literal: true

source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby '3.0.2'

gem 'rails', '~> 6.1.3', '>= 6.1.3.2'

gem 'sqlite3', '~> 1.4'

gem 'puma', '~> 5.0'

gem 'webpacker', '6.0.0.rc.6'

gem 'bcrypt', '~> 3.1.7'

gem 'activerecord-import', '~> 1.2'
gem 'blueprinter', '~> 0.25'
gem 'caxlsx', '~> 3.1'
gem 'caxlsx_rails', '~> 0.6'
gem 'dotenv-rails', '~> 2.7'
gem 'draper', '~> 4.0'
gem 'i18n-tasks', '~> 0.9.34'
gem 'lokalise_rails', '~> 3'
gem 'pagy', '~> 5.0'
gem 'pundit', '~> 2.1'
gem 'rails-i18n', '~> 6'
gem 'rubyXL', '~> 3.4'
gem 'rubyzip', '~> 2'
gem 'sidekiq', '~> 6'
gem 'valid_email2', '~> 4.0'

# Use Active Storage variant
# gem 'image_processing', '~> 1.2'

# Reduces boot times through caching; required in config/boot.rb
gem 'bootsnap', '>= 1.4.4', require: false

group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get a
debugger console
  gem 'byebug', platforms: %i[mri mingw x64_mingw]
  gem 'faker', '~> 2'
  gem 'pry-rails'
end
```

```
group :development do
  # Access an interactive console on exception pages or by calling
'console' anywhere in the code.
  gem 'web-console', '>= 4.1.0'
   # Display performance information such as SQL time and flame
graphs for each request in your browser.
   # Can be configured to work on production as well see:
https://github.com/MiniProfiler/rack-mini-
profiler/blob/master/README.md
  gem 'bullet'
  gem 'letter_opener'
  gem 'rack-mini-profiler', '~> 2.0'
  gem 'rubocop', '~> 1.18', require: false
  gem 'rubocop-i18n', '~> 3', require: false
  gem 'rubocop-performance', '~> 1.11', require: false
  gem 'rubocop-rails', '~> 2.11', require: false
end

# Windows does not include zoneinfo files, so bundle the tzinfo-
data gem
gem 'tzinfo-data', platforms: %i[mingw mswin x64_mingw jruby]
```

Лістинг файлу package.json

```
  {
  "name": "ask-it",
  "private": true,
  "dependencies": {
    "@popperjs/core": "^2.9.2",
    "@rails/actioncable": "^6.0.0",
    "@rails/activestorage": "^6.0.0",
    "@rails/ujs": "^6.0.0",
    "@rails/webpacker": "6.0.0-rc.6",
    "autoprefixer": "^10.3.1",
    "bootstrap": "^5.1.0",
    "css-loader": "^6.2.0",
    "css-minimizer-webpack-plugin": "^3.0.2",
    "mini-css-extract-plugin": "^2.2.0",
    "postcss": "^8.3.6",
    "postcss-flexbugs-fixes": "^5.0.2",
    "postcss-import": "^14.0.2",
    "postcss-preset-env": "^6.7.0",
    "sass": "^1.37.5",
    "sass-loader": "^12.1.0",
    "tom-select": "^2.0.0-rc.4",
    "turbolinks": "^5.2.0",
    "webpack": "5.62.1",
    "webpack-cli": "4.9.1"
  },
  "version": "0.1.0",
  "devDependencies": {
    "@webpack-cli/serve": "^1.5.2",
    "webpack-dev-server": "^4.0.0"
```

```
  },
  "babel": {
    "presets": [
      "./node_modules/@rails/webpacker/package/babel/preset.js"
    ]
  },
  "browserslist": [
    "defaults"
  ]
}
```