РЕФЕРАТ

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-43, 2023 рік. Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 62 с., 14 рис., 5 табл., 2 додатків.

Об'єкт дослідження – клієнт-серверна інформаційна системи для IT-компанії.

Мета роботи – Спроектувати систему обслуговування заявок з використанням UML, IDEF0, DFD діаграм.

Метод дослідження – аналіз предметної галузі з погляду проблеми обслуговування заявок та засобів розробки необхідного програмного забезпечення.

Отримані результати – розроблено програмний засіб, що дозволяє підтримувати процеси обслуговування заявок у межах обраної предметної галузі.

ANOTATION

Qualification work for obtaining an educational degree "bachelor" in specialty 121 - Engineering of program support. Ternopil National Technical University. Ivan Puluj, Faculty of Computer and Information Systems and Gram Engineering, Department of Gram Engineering, group CPs-43, 2023. The explanatory note to the qualification work for the educational degree "bachelor" contains: 62 pages 14 pictures, 5 tables, 2 addition.

The object of the research – a client-server information system for an IT company.

The purpose of the work – design an application service system using UML, IDEF0, DFD diagrams.

The research method is an analysis of the subject area from the point of view of the problem of servicing applications and means of developing the necessary software.

The obtained results – a software tool was developed that allows you to support the processes of service of applications within the selected subject area.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IT – Information Technology. Інформаційні технології;
HTML – HyperText Markup Language. Мова розмітки веб сторінок
СУБД – Database Management System. Набір програм та комплексів для взаємопов’язання програм та доступу до них;
CSS – Cascading Style Sheets. Мова стилів, для задання візуального виду документа, написаному на HTML
JSP – JavaServer Pages. Використовується для динамічної генерації HTML та XML веб-сторінок;
API – Application Programming Interface. Програмний набір протоколів та інструментів для створення програмного забезпечення;
XML – eXtensible Markup Language. Розширена мова розмітки.

ВСТУП

Розвиток інформаційного суспільства є одним із національних пріоритетів України і розглядається як загальнонаціональне завдання, що потребує координації та об'єднання зусиль держави, бізнесу та громадянського суспільства. 23 березня 2016 р. Постановою Кабінету Міністрів України №235 затверджено Державну програму розвитку цифрової економіки та інформаційного суспільства на 2016-2022 роки. Вона включає 3 підпрограми:

1. Інформаційно-комунікаційна інфраструктура.
2. Інфраструктура інформації.
3. Цифрова трансформація.

Метою Державної програми є вдосконалення умов, що сприяють трансформації сфер людської діяльності під впливом інформаційно-комунікаційних технологій, включаючи формування цифрової економіки, розвиток інформаційного суспільства та вдосконалення електронного уряду [1].

Відповідно до другої підпрограми одним із провідних напрямів розвитку інфраструктури інформатизації є забезпечення безперервності, безвідмовності, безпеки інформаційних потоків. Подальше вдосконалення національної інформаційно-комунікаційної інфраструктури є одним із найважливіших державних завдань нашої держави відповідно до Стратегії інформатизації України на 2016-2022 роки [3].

Зі збільшенням кількості інтернет-користувачів, розвитком інформаційної інфраструктури, ускладненням програмно-апаратного забезпечення та підвищенням вимог до якості обслуговування виникла потреба в автоматизації управління заявками внутрішніх та зовнішніх клієнтів. Служба підтримки користувачів Help Desk, що з'явилася більше 20 років тому, виступає єдиною точкою контактів між постачальником послуг та численними користувачами. В даний час типова служба підтримки

користувачів керує інцидентами, запитами на зміну, запитами на обслуговування та здійснює комунікації з користувачами. Під інцидентом розуміється незапланованe переривання чи зниження якості IT-послуги. Управління інцидентами забезпечує мінімізацію впливу на бізнес та відновлення нормального функціонування послуги найшвидшим способом.

З програмними продуктами системи працюють спеціалісти IT-відділів у галузі реагування на проблеми надання IT-послуг; адміністративно-господарського відділу з питань матеріально-технічного забезпечення; відділу кадрів у сфері підбору персоналу; відділу документаційного забезпечення оформлення договірної роботи.

З погляду керівників система Help Desk дозволяє:

1. Організувати оперативне отримання стандартних послуг.
2. Зменшити їхню вартість для бізнесу.
3. Підвищити рівень контролю за якісними та кількісними показниками сервісів, що надаються.

Завданнями її застосування виступають:

1. Збільшення доступності послуг для кінцевих користувачів.
2. Поліпшення якості послуг, що надаються споживачеві та його задоволеність послугами.
3. Поліпшення комунікації та взаємодії.
4. Зниження впливу інцидентів на бізнес.

Система обслуговування заявок – основний компонент Help Desk систем. Отже, тема дослідження є актуальною.

В результаті дипломного дослідження вирішено низку завдань:

1. Здійснено аналіз предметної галузі з погляду проблеми обслуговування заявок та засобів розробки необхідного програмного забезпечення.
2. Дано тлумачення основних понять у цих галузях з використанням існуючих стандартів.
3. Спроектовано систему обслуговування заявок з використанням UML,

IDEF0, DFD діаграм.

4. Розроблено програмний засіб, що дозволяє підтримувати процеси обслуговування заявок у межах обраної предметної галузі.

Дипломну роботу виконано на даних компанії ТОВ «ЛВО». Ця компанія спеціалізується на розробці IT-проектів для банків, державних та комерційних підприємств. Замовниками є організації України у, а також країни ЄАЕС. ТОВ «ЛВО» займає 4 місце серед найбільших постачальників програмного забезпечення на внутрішньому ринку. Компанія є резидентом Парку високих технологій та членом науково-технічної асоціації «ІнфоПарк».

Дипломна робота складається з 3 розділів, вступу та висновків. У першому розділі наведено основні визначення з погляду систем обслуговування заявок, методів та засобів розробки програмного забезпечення; проведено аналіз існуючих систем обслуговування заявок, методів та засобів розробки програмного забезпечення. У другому розділі виконується проектування системи з допомогою UML, IDEF0, DFD діаграм. А також описуються її функції та засоби розробки даної системи. Третій розділ визначає процес розробки програмного засобу системи обслуговування заявок.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Основні поняття

1.1.1. Визначення

"System - set of interrelated or interacting elements" [10].

Система - сукупність взаємозалежних та взаємодіючих елементів.

"Process - set of interrelated or interacting activities that use inputs to deliver an intended result" [10].

Процес - сукупність взаємопов'язаних та взаємодіючих видів діяльності, які використовують вхідні дані для досягнення бажаного результату.

"Requirement - need or expectation that is stated, generally implied or obligatory" [10].

Вимога — потреба чи очікування, яке встановлено, зазвичай передбачається чи обов'язковим.

"Objective - result to be achieved" [10].

Мета - результат, який має бути досягнутий.

«Project - unique process, consisting of set of coordinated and controlled activities with start and finish datas, undertaken to achieve an objective conforming to specific requirements, including the constraints of time, cost and resources» [10].

Проект — унікальний процес, що складається з сукупності скоординованих та керованих видів діяльності з початковою та кінцевою датами, вжитий для досягнення мети, що відповідає конкретним вимогам, що включає обмеження щодо термінів, вартості та ресурсів.

«Validation - confirmation, through the provision of objective evidence, that therequirements for specific intended use or application have been fulfilled» [10].

Валідація – підтвердження, що вимоги для конкретного передбачуваного використання чи застосування були виконані шляхом надання об'єктивних доказів.

«Localization — створення of national or specific regional version of a product» [18].

Локалізація — створення національної чи специфічної регіональної версії продукту.

«Internationalization — процес розвитку інформаційного процесу, що це є сутильним для an international audience» [18].

Інтернаціоналізація – процес розробки інформації так, щоб вона підходила міжнародній аудиторії.

«Design and development — set of processes that transform requirements for an object into more detailed requirements for that object» [10].

Проектування та розробка - набір процесів, які перетворюють вимоги для об'єкта на більш детальні вимоги.

«Incident — непланована interruption на службу, зменшення в якості послуги або на те, що не може бути спричинена послугою для customer or user» [7].

Інцидент – незапланованe переривання у наданні послуги, зниження її якості чи подія, яка ще не вплинула на послугу для замовника чи користувача.

«Record - document stating results achieved or providing evidence of activities performed» [7].

Запис - документ, що містить досягнуті результати або свідоцтва здійсненої діяльності.

«Service request — request for information, advice, access to a service or a pre- approved change» [7].

Запит на обслуговування — запит на надання інформації, консультації, доступу до послуги або запит на попередньо затверджену зміну

«Problem - cause of one or more actual or potential incidents» [7].

Проблема — причина одного чи кількох актуальних чи можливих інцидентів.

«User — окрема або група that interacts with or benefits from aservice or services» [7].

Користувач — це людина або група, яка взаємодіє з послугою або послугами або користується ними.

"Information security - preservation of confidentiality, integrity and availability of information" [7].

Інформаційна безпека – збереження конфіденційності, цілісності та можливості доступу до інформації.

«Service - means of delivering value for the customer by faciliting outcomes

the customer wants to achieve» [7].

Послуга – спосіб надання цінності замовнику через сприяння йому в отриманні кінцевих результатів, яких замовник хоче досягти.

«Теорія СМО (систем масового обслуговування) - область прикладної математики, що займається аналізом процесів у системах виробництва, обслуговування, управління, в яких однорідні події повторюються багаторазово, наприклад, на підприємствах побутового обслуговування; у системах прийому, переробки та передачі інформації; автоматичних лініях виробництва та ін» [2].

«Обслуговуючі пристрої, канали обслуговування-засоби, обслуговуючі вимоги (заявки)» [2].

"Абсолютна пропускна спроможність системи - середня кількість заявок, що обслуговуються в одиницю часу" [2].

«Відносна пропускна спроможність - середня частка заявок, що надійшли, обслуговуються системою» [2].

«Вірогідність відмови системи - ймовірність того, що заявка залишить СМО не обслуженою» [2].

"Design, verb - <process> to define the architecture, system elements, interfaces, and other characteristics of a system or system element" [17].

"Design, noun - result of the process in design, verb" [17].

Розробка — визначення архітектури, елементів системи, інтерфейсів та інших характеристик системи або елементів системи.

«Software maintenance — це повна діяльність, що потребує забезпечення ефективної підтримки до програмного забезпечення. Activities є performed during the pre-delivery stage as well as the post-delivery stage» [13].

Обслуговування програмного забезпечення — сукупність дій, необхідні забезпечення економічно ефективної підтримки системи програмного забезпечення. Дії виконуються на етапі перед здаванням, а також на етапі після здавання системи програмного забезпечення.

«Life cycle — evolution of a system, product, service, project or other human-

made entity from conception through retirement» [19].

Життєвий цикл — еволюція системи, продукту, послуги, проекту чи іншого створеного людиною об'єкта від концепції до вилучення з експлуатації.

1.1.2. Історія

Метод у контексті засобів розробки програмного забезпечення – це набір технік та правил.

Під засобом розробки програмного забезпечення розуміється набір методів та методик, прийомів та інструментальних програм для створення програмного коду.

На різних етапах розробки програмного забезпечення використовуються різні засоби та методи. Для того, щоб систематизувати

блок-схеми, UML (Unified Modeling) діаграми, DFD (Data Flow Diagrams) діаграми та IDEF0 діаграми. Також можна використовувати BPMN (Business Process Model and Notation) діаграми.

IDEF0 діаграми описують систему, починаючи від виявленої точки зору до конкретної мети. Ці діаграми розташовуються різних рівнях. На верхньому рівні є найбільш загальна діаграма, а кожен наступний рівень може включати до 6 діаграм, які деталізують діаграму на верхньому рівні. Діаграма складається з блоків та дуг. Блоки діаграми пронумеровані від 1 до 6. Верхня діаграма має назву A0. Кожен процес на діаграмі A0 може бути розкладений на підпроцеси та змодельований на діаграмі нижнього рівня, яка матиме назву A1-A6 залежно від номера обраного блоку. Кожен процес із діаграм A1-A6 може бути розкладений на підпроцес і змодельований на діаграмах A16-A66, де перша цифра відповідає за номер вищестоящої діаграми, друга
- За номер процесу. Блоки є дією, а дуги визначають взаємозв'язок блоків. Ліва сторона блоку призначена для вхідних дуг, права - для вихідних, верхня - для дуг управління, нижня - для механізмів.

Другий етап стадії проектування здійснюють за допомогою діаграм класів та пакетів.

Для створення UML діаграм можна використовувати Enterprise Architect — програмне забезпечення від австралійської компанії Sparx Systems, розроблене з метою управління бізнес-проектами та побудови бізнес-планів.

Розробка макетів інтерфейсу користувача допомагає зрозуміти, як будуть виглядати вікна, сторінки в програмі, що розробляється. Ця стадія здійснюється із застосуванням таких засобів як Figma або Axure. Figma - кросплатформний сервіс для прототипування та розробки інтерфейсів. Figma дозволяє одночасно працювати над одним і тим самим проектом кільком людям.

На вибір засобів реалізації програмного продукту впливають підходи, обрані на етапі проектування. Проте, можна назвати основні види коштів. По-перше, це мови програмування. Розглянемо мови для створення веб-застосунків.

1. Java - сильно типізована об'єктно-орієнтована мова програмування. Ця мова була розроблена компанією Sun Microsystems, яку потім придбала компанія Oracle. Програми java є кросплатформовими, тому що вони транслюються в спеціальний байт-код і можуть виконуватися на будь-якій архітектурі за допомогою віртуальної java-машини. На даний момент Java одна з найпопулярніших мов програмування. Завдяки безлічі доповнень і бібліотек, що підключаються, він ідеально підходить для вирішення поставленого завдання.

2. PHP — мова, за допомогою якої написано більшість сайтів та веб-сервісів. Приклад сайтів написаних на PHP: facebook.com, vk.com, baidu.com. Ця мова приваблює багатьох фахівців своєю простотою.

3. Python — інтерпретована мова програмування. Простий у використанні та гнучкий. Містить невелику кількість ключових слів. Інтерпретатор Python реалізований на багатьох платформах та операційних системах.

4. NodeJS - програмна платформа, яка заснована на програмному засобі (движку) V8[1]він служить для трансляції JavaScript в машинний код. NodeJS дозволяє JavaScript взаємодіяти з пристроями вводу-виводу, підключати зовнішні бібліотеки, які можуть бути написані різними мовами. NodeJS в основному використовується на сервері та виконує роль веб-сервера. Однак він дозволяє розробляти і десктопні віконні програми та програмувати мікроконтролери. Основу NodeJS складають орієнтоване та асинхронне (або реактивне) програмування з неблокуючим введенням/виводом.

Другий вид засобів реалізації – це системи управління базами даних. Вибір системи управління базою даних є важливою частиною будь-якого проекту. Від цього вибору істотно залежить швидкість роботи програми. Відповідно це позначиться на рівні задоволеності користувача. Розглянемо найпоширеніші бази даних:

1. MySQL є популярною повноцінною системою управління базами даних. Вона відмінно підходить для роботи з різними програмами. Для цієї СУБД існує велика кількість плагінів, які дозволяють полегшити роботу із системою. Також MySQL добре документована. MySQL забезпечує безпечний доступ до даних. Недоліками цієї СУБД є повільна розробка проекту та слабка стійкість до відмови.

2. PostgreSQL – реляційна СУБД з відкритим вихідним кодом. Вона є професійнішою СУБД, ніж MySQL. У PostgreSQL велика продуктивність, і дана СУБД легко розширюється за допомогою процедур, що зберігаються. Як і MySQL, вона надає безпечний доступ до даних.

3. MongoDB є базою даних NoSQL. Вона не структурована та має хорошу пропускну здатність. У MongoDB відсутні класичні транзакції, які є у MySql та PostgreSQL.

Третій засіб – це системи управління версіями. Існують такі системи: Git, SVN (Subversion), Mercurial.

На стадії тестування відбувається випробування програмного продукту на відповідність вимогам, виявляються помилки, покращується якість

програмного продукту. Серед засобів тестування програмного забезпечення виділяють кошти для аналізу коду, тестування функціональності та продуктивності.

Останнім часом особливою популярністю користуються веб-програми. Веб-додаток є клієнт-серверним додатком. Клієнт взаємодіє із сервером за допомогою браузера. Сервер приймає запити HTTP від клієнтів і видає їм HTTP відповіді. Логіка розподіляється між сервером та клієнтом. Зберігання інформації в базі даних.

Переваги веб-додатків такі:

1. Встановлення веб-додатків дешевше та простіше. Знижуються витрати на утримання відділів, які відповідають за встановлення та супровід ПЗ.
2. Оновлення веб-додатків дешевше та простіше. Щоб оновити веб-програму, достатньо її оновити лише на сервері.
3. Веб-додатки універсальні. Вони можуть працювати на будь-комупристрої, під керуванням будь-якої ОС та у будь-якому браузері.

Для багатьох проектів характерна рольова система. Рольова система використовується у таких випадках:

1. Фрагменти системи потребують певних прав доступу.
2. Повинні бути організовані групи, члени яких повинні бути наділені особливими правами.
3. Користувачі однієї групи повинні мати можливість керувати правами користувачів іншої групи.
4. Система має мати зрозумілу реалізацію.
5. Потрібно вести запис дій користувача до журналу. Повинний бути безпосередній зв'язок між системою прав доступу та системою запису до журналу.

1.2. Клієнт-серверна архітектура стосовно БД.

1.2.1. Поняття архітектури клієнт-сервер

Одне із завдань IT (Information Technology) відділів — відповідати на заявки, які можуть надходити у будь-який час і від будь-якої людини. Тому має бути створена система обслуговування заявок, що дозволяє забезпечити швидке та надійне вирішення проблем клієнтів.

Перед системами обслуговування заявок висуваються такі вимоги:

1. забезпечити єдину точку контакту між користувачами та послугами;
2. керувати інцидентами та запитами;
3. керувати комунікаціями з користувачами: інформувати користувача про хід виконання його запиту, гарантувати реєстрацію та збереження кожного запиту користувача;
4. брати участь у якнайшвидшому відновленні роботи сервісів.

В даний час для комунікації з клієнтами існує безліч цифрових каналів: веб-сайти, електронна пошта, чат в режимі реального часу, мобільні програми, обмін текстовими повідомленнями, онлайн-форуми та соціальні мережі. Використання цифрових каналів входить у щоденну практику більшості бізнес-структур, що дозволяє організувати самообслуговування та знизити навантаження на call-центри. За даними опитування, проведеного The Boston Consulting Group і NICE, 82% споживачів у вибірці з країн із розвиненою економікою використовують веб-канали самообслуговування, 46% використовують мобільні програми. Проте, те саме опитування показало, що 82% споживачів, як і раніше, дзвонять у контакт-центри, щоб особисто поспілкуватися з агентами [11]. Як єдина точка контакту в системах обслуговування заявок може виступати єдиний номер.

Управління інцидентами забезпечує мінімізацію впливу на бізнес, відновлення нормального функціонування послуги найшвидшим способом,

підтримку рівня якості та доступності сервісів. Інциденти можуть повторюватися, якщо не усунути причину їх виникнення – проблему. Управління проблемами допомагає запобігти виникненню інцидентів та мінімізувати вплив тих інцидентів, яких не можна запобігти. З погляду бізнесу управління проблемами підвищує доступність IT-послуг та продуктивність персоналу, скорочує витрати на рішення інцидентів.

Під запитом на обслуговування розуміється запит щодо надання інформації, консультації, доступу до послуги. Управління запитами на обслуговування

бюрократію у вимогах та отримання доступу до послуг, збільшити рівень контролю над необхідними послугами через централізовану функцію управління.

На даний момент існує безліч готових програмних продуктів Help Desk. Проаналізуємо функціональні можливості та особливості деяких із них. Для аналізу було обрано такі системи: Spiceworks, Happydesk, Freshdesk, Open-source Ticket Request System. Розглянемо кожну систему та зробимо їхній порівняльний аналіз.

1. Spiceworks.

Spiceworks являє собою безкоштовний простий у встановленні та використанні Help Desk.

Користувачі можуть надсилати заявки до Spiceworks через спеціальний портал користувача. Цей портал інтегрований з Outlook, що дозволяє користувачеві швидко реєструватися. У свою чергу, зі списку всіх заявок виконавці обирають заявку для обробки. Список заявок у Spiceworks зручний тим, що не треба відкривати заявку, щоб переглянути всю інформацію по ній, достатньо навести на неї курсор.

Spiceworks надає базу знань, за допомогою якої клієнт може знайти вирішення свого завдання, не надсилаючи заявки до системи.

Також дана система дозволяє створювати звіти за встановлений час та експортувати їх у файл формату xls. Для відображення статистики заявок є

інформаційна панель для представлення згрупованих даних.

Spiceworks має мобільну версію для зручнішої роботи із заявками.

Інтерфейс Spiceworks представлений рисунку 1.1

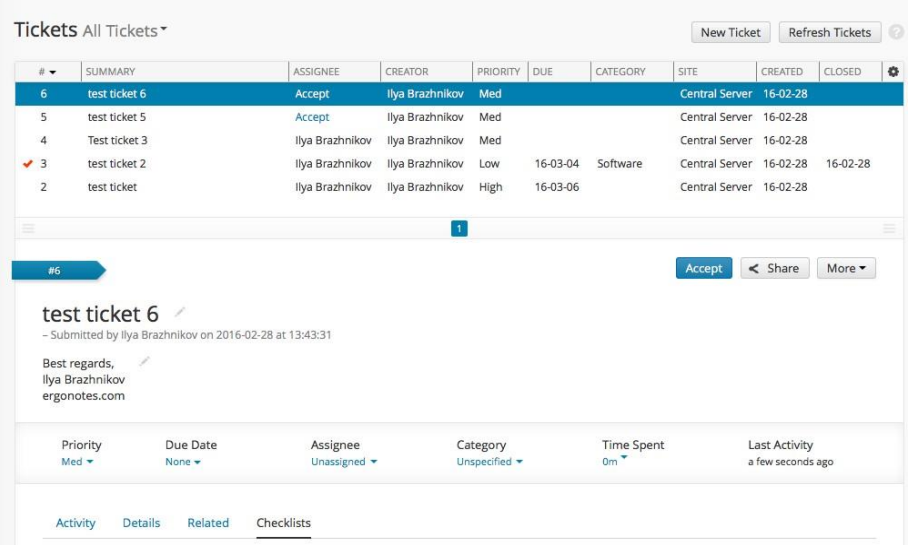


Рисунок 1.1 - Інтерфейс Spiceworks

1.2.2. Двохрівнева клієнт-серверна архітектура.

Розробники Happydesk позиціонують її як інтуїтивно зрозумілу систему. Функції цієї системи: збирання, облік, обробка та аналіз запитів від клієнтів. Happydesk є гнучкою багатоканальною системою. Це означає, що запити від клієнтів можуть надходити через електронну пошту, програми обміну повідомленнями або спеціальний віджет[2]на сайті.

Існують правила направлення заявок. Заявки розподіляються виконавцям, враховуючи завантаженість кожного виконавця та його досвід у виконанні подібних заявок.

Для виконавців Happydesk має такі можливості, як заготовлені відповіді, які виконавець може додавати та використовувати; перегляд повної історії заявок від клієнтів; складний фільтр, що налаштовується за заявками; керування пріоритетом повідомлень.

Клієнт може самостійно знайти вирішення своєї проблеми за допомогою зручного пошуку на базі знань. Завдяки цьому кількість листів до системи підтримки скорочується на 64% [6]. Крім цього, Happydesk надає можливість клієнтам оцінювати задоволеність роботою служби. Також можна переглядати статистику якості роботи окремих працівників чи відділів.

Happydesk інтегруємо у існуючі системи підтримки клієнтів. Ця система підтримує делегування запитів між співробітниками, розмежування прав адміністрування. Контроль ефективності роботи працівників здійснюється за допомогою графіків та зведених таблиць.

У безкоштовній версії Happydesk має обмежену кількість можливостей. Зокрема, безкоштовна версія може підтримувати трохи більше трьох співробітників; в ній неможливий автоматичний розподіл заявок, завдання правил та шаблонів для виконавців.

Інтерфейс Happydesk представлений рисунку 1.2.

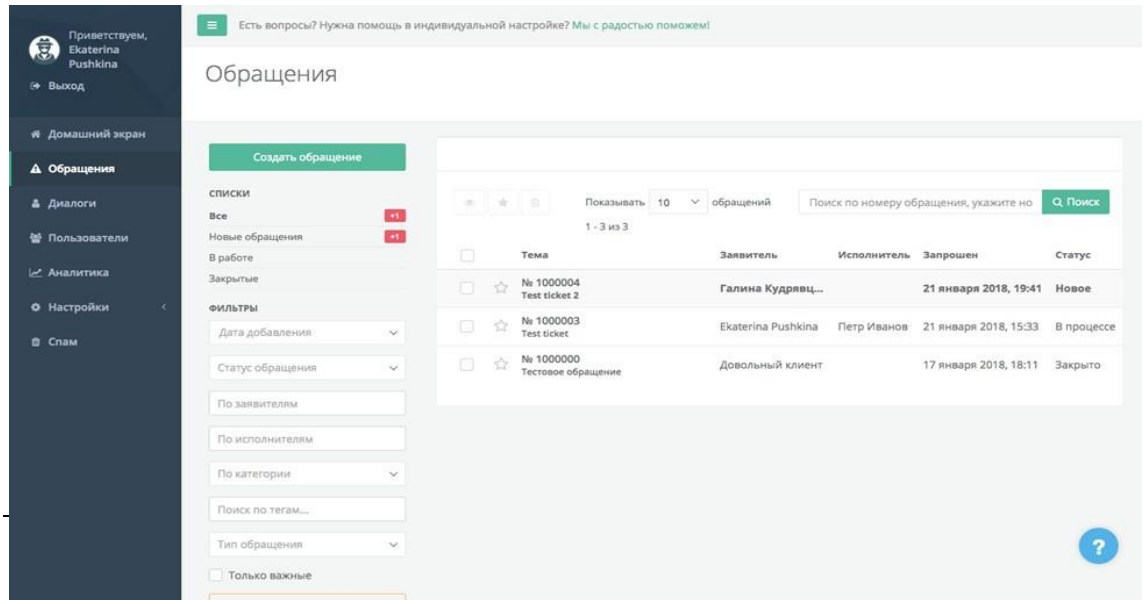


Рисунок 1.2 - Інтерфейс Happydesk

1.2.3. Багаторівнева архітектура клієнт-сервер (Multitier architecture)

Freshdesk є багатоканальною системою. Він інтегрований із Twitter, Facebook. Дана система має спеціальний віджет для надсилання заявок.

Зібрані заявки надходять до спільної поштової скриньки. Це рішення дозволяє виконавцям обробляти заявки, не заважаючи один одному. Повідомлення, яке надходить на адресу електронної пошти служби, стає заявкою. Далі виконавцем на кожну заявку може бути пріоритет і категорія. Виконавець може делегувати заявку члену команди.

Завдяки базі знань та клієнтським форумам клієнт отримує можливість вирішувати проблеми, не надсилаючи заявок до системи. База знань допомагає клієнтам, а також скорочує обсяг заявок, що входять. Додатковою функцією форумів є збір відгуків щодо якості роботи системи. Зовнішній вигляд системи може змінюватись.

Freshdesk інтегруємо із CRM системами. Цей Help Desk має програми, сумісні з OCiOSта Android, що дозволяє робити підтримку клієнтів мобільного.

Для відстеження продуктивності Freshdesk пропонує використовувати звіти. Це можуть бути базові звіти, зведені звіти та аналіз навантаження.

Відмінною рисою Freshdesk є гейміфікація[3]. Вона служить мотивації виконавців. Кожному виконавцю нараховується певна кількість балів залежно від часу виконання заявки та оцінки якості виконаної роботи від клієнта. Нараховані бали можуть забиратися у разі недотримання термінів або надходження негативного відкликання від клієнта. Ці бали бачить керівник та може залежно від них регулювати розмір премії виконавця. Також є обов'язкові для всіх виконавців «квести». Наприклад, «квест[4]» може бути таким: «закрити 10 заявок на тиждень».

Безкоштовна версія надає лише базу знань та підтримку на адресу електронної пошти.

Інтерфейс Freshdesk представлений рисунку 1.3.

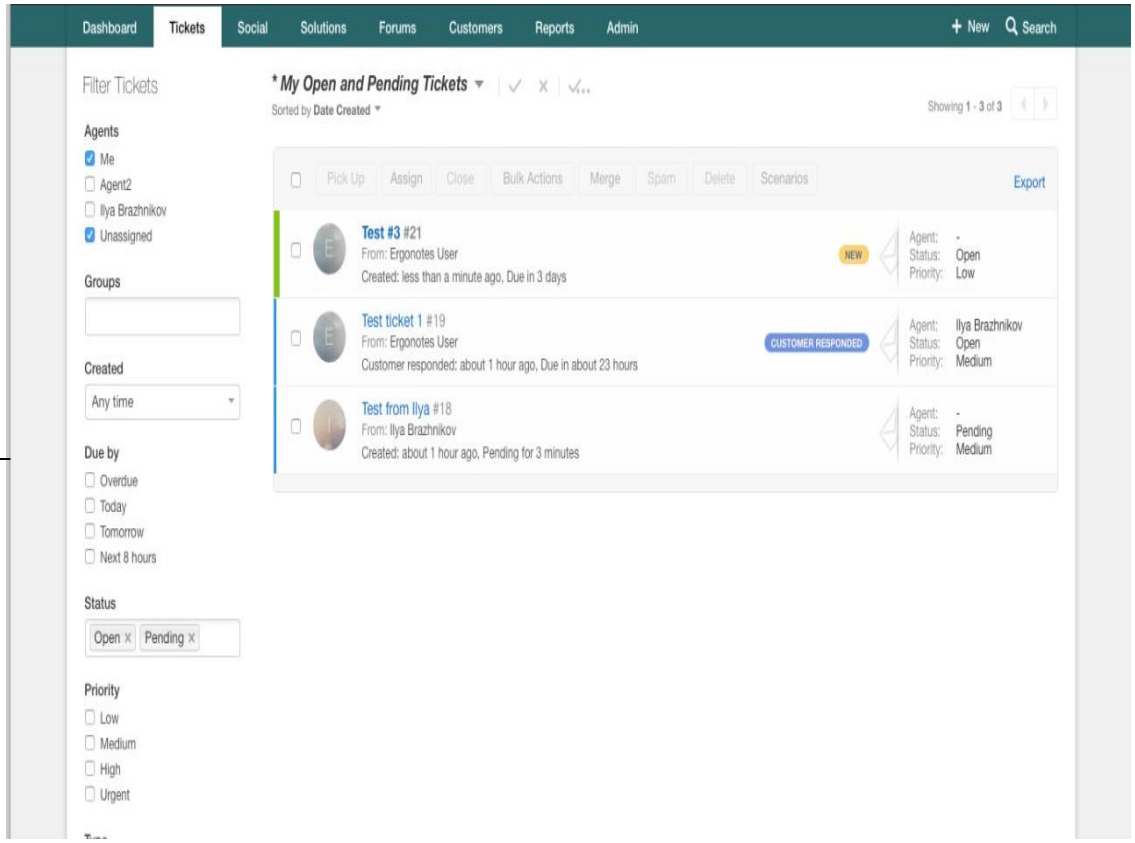


Рисунок 1.3 - Інтерфейс Freshde

1.2.4. Моделі клієнт-сервер

OTSR – відкрита система обробки заявок. Історія цього інструменту розпочинається з 2001 року. OTSR надає різні інтерфейси для клієнтів та виконавців. Дизайн інтерфейсів можна налаштувати за власними уподобаннями.

Є можливість створювати звіти за допомогою розширеного генератора звітів за статистичними даними і параметрами планування, що налаштовуються.

Відмінною особливістю даної системи є те, що вона підтримує більше 30 мов та різні часові пояси, а також має фільтрацію для заявок, що надходять.

OTSR підтримує різні системи управління базами даних: MySQL 5.0 та вище, PostgreSQL 8.4 і вище, Oracle 10g; сервера Apache2+mod_perl2 або вище, Веб-сервер з підтримкою CGI-інтерфейсу; браузери: Google Chrome, Firefox version 10 та вище, Safari version 5 і вище.

Інтерфейс OTSR представлений рисунку 1.4.

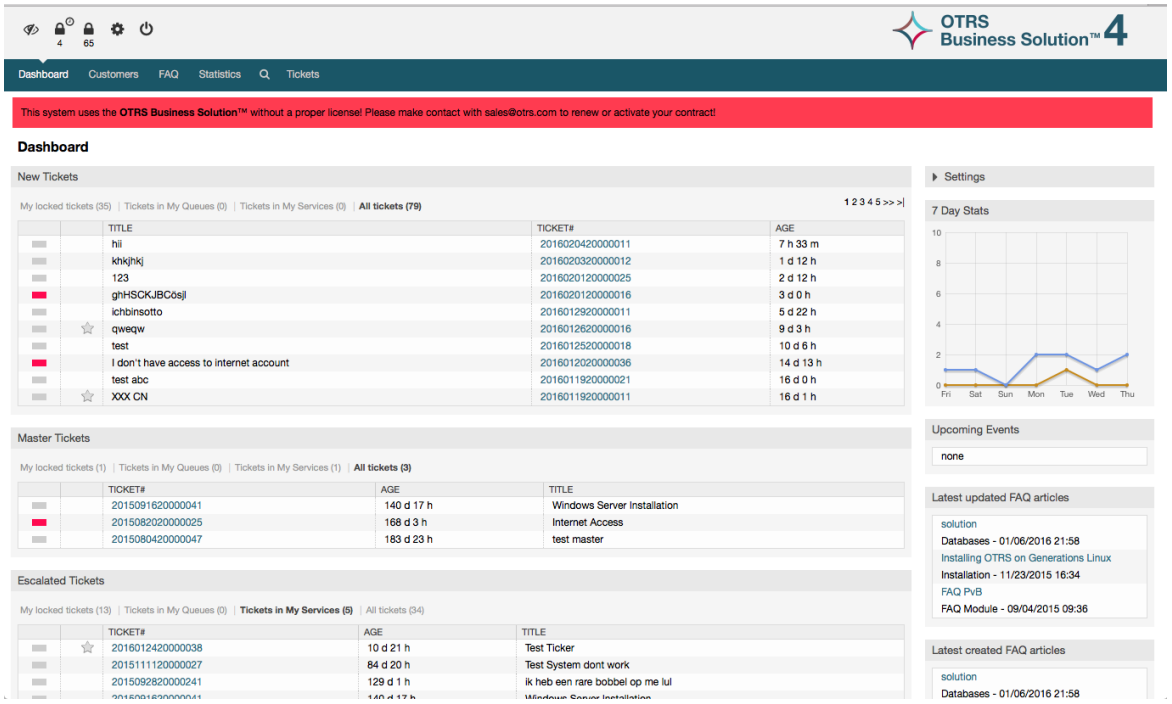


Рисунок 1.4 - Інтерфейс OTSR

У таблиці 1.1 представлені переваги та недоліки розглянутих систем.

Таблиця 1.1 - Аналіз систем Help Desk

Help Desk	Переваги	Недоліки
Spiceworks	1. безкоштовний; 2. простий у встановленні; 3. інтегрований зOutlook; 4. зручний перелікзаявок; 5. наявність бази знань; 6. експорт звітів у xls файл; 7. наявність мобільноюверсії.	1. облікові записи прив'язуються до сервера Spiceworks; 2. відсутні правила автоматизації; 3. показується реклама інших товарів; 4. лише для ОС MS Windows.
Happydesk	1. багатоканальна система; 2. заготовленіша блони; 3. пріоритезація заявок; 4. є правила направлення заявок; 5. наявність бази знань; 6. інтегрується у існуючі системи підтримки; 7. можливість побудови графіків та таблиць.	1. безкоштовна версія має обмежені можливості; 2. немає мобільної версії для iOS.
Freshdesk	1. багатоканальна система; 2. наявність бази знань; 3. можливість редагування зовнішнього вигляду системи; 4. інтегруємо із CRM системами;	1. немає правил керування заявками; 2. Безкоштовна версія має обмежені можливості.

Таблиця 1.2 - Порівняння систем Help Desk

	Spiceworks	Happy desk	Freshdesk	OTSR	Lwo Help Desk
багатоканальність	-	+	+	-	-
база знань	+	+	+	+	-
генератор звітів	+	+	+	+	+
безкоштовність	+	-	-	+	-
правила направлення заявок	+	+	+	-	-
наявність мобільної версії	+	+-	+	-	-
мультимовний	-	-	-	+	+

2. ПРОЕКТНА ЧАСТИНА

2.1. Клієнт-серверна архітектура стосовно ІС

Специфікація функцій, які повинна виконувати система обслуговування заявок, подана у таблиці 2.1.

Таблиця 2.1 - Специфікація функцій проекту

Назва функції	Опис функції
Авторизація у системі	Введення логіну та пароля у відповідні поля форми для ідентифікації в системі з певною роллю.
Реєстрація в системі	Введення імені, електронної пошти та пароля у відповідні поля форми.
Вихід з системи	Вихід із системи з перенаправленням на екран авторизації.
Видалення користувача	Видалення запису у базі даних про конкретного користувача адміністратором.
Редагування списку користувачів	Редагування всіх полів запису в базі даних конкретного користувача адміністратором.
Призначення прав	Редагування прав зареєстрованого користувача адміністратором.
Подача заявки	Створення нового запису у системі з описом інциденту користувачем.
Видалення заявки	Видалення заявки користувачем у тому випадку, якщо заявка ще не була призначена виконавцю.
Вибір виконавця	Перенаправлення заявки конкретному виконавцю оператором.
Зміна стану заявки	Зміна поточного стану заявки.
Додавання організації	Додавання запису до бази даних адміністратором організації.

Таблиця 2.1 - продовження.

Назва функції	Опис функції
Редагування списку організацій	Редагування всіх полів запису у базі даних про конкретну організацію адміністратором.
Видалення організації	Видалення запису у базі даних про конкретну організацію адміністратором.
Додавання відділу	Додавання запису до бази даних адміністратором про відділ.
Редагуванняспи ску відділів	Редагування всіх полів запису в базі даних конкретного відділу адміністратором.
Видалення відділу	Видалення запису в базі даних конкретного відділу адміністратором.
Зміна теми	Зміна теми оформлення системи.
Зміна мови	Зміна мови системи.

2.2. Клієнт-серверні обчислення

2.2.1. Піраміда моделі «клієнт-сервер»

Серверна частина програми реалізована за допомогою бібліотеки Spring

1) створення та налаштування додатків.

Вхідною точкою програми є клас з методом main, позначений інструкцією @SpringBootApplication. Ця інструкція включає наступні інструкції: @EnableAutoConfiguration, @ComponentScan, @SpringBootConfiguration.

@EnableAutoConfiguration анотація дозволяє автоматично конфігурувати програму на основі jar-залежностей, доданих до проекту. @ComponentScan анотація потрібна для сканування всіх класів та пакетів під час ініціалізації програми. Boot, клієнтська - Angular.

Spring Boot — бібліотека з відкритим кодом для створення мікросервісів[5], що базується на мові Java. Ця бібліотека використовується для написання повноцінних Spring-додатків. Spring Boot має такі цілі:

2) уникнути складності конфігурації xml у Spring;
3) спростити розробку готових до запуску Spring додатків;

надати простий спосіб

З метою уникнути складного процесу управління залежностями Spring Boot ввели starter пакети. Вони є набором дескрипторів залежностей, які легко додати до програми. Наприклад, якщо з'являється необхідність використовувати Spring JPA для доступу до бази даних у проекті, достатньо прописати spring-boot-starter-data-jpa залежність у файлі pom.xml. Основні пакети starter наступні:

1) spring-boot-starter-actuator - використовується для моніторингу та управління додатком;
2) spring-boot-starter-security - використовується для Spring Security, що відповідає за авторизацію та аутентифікацію користувачів та доступ до даних;
3) spring-boot-starter-web – використовується для створення REST (Representational state transfer) архітектури;
4) spring-boot-starter-test – використовується для написання тест-кейсів;
5) spring-boot-starter-data-jpa — використовується для доступу до даних, наприклад, реляційних і нереляційних баз даних;

Spring Boot веб-додаток включає вбудований веб-сервер, наприклад Apache Tomcat, Jetty або JBoss Undertow.

Angular — бібліотека компанії Google, призначена в основному для розробки SPA (single page application) додатків.

Бібліотека адаптує та розширює HTML за допомогою директив, а також синхронізує модель та подання за рахунок різного виду прив'язок даних.

Плюси Angular:

1. Angular поділяє DOM-маніпуляції та логіку програми. Це допомагає зробити процес розробки та тестування простіше.
2. Angular добре працює з бібліотеками Karma та Jasmin, які призначені для тестування.

3. Angular дотримується MVC (Model View Controller) шаблону проектування та заохочує слабкий зв'язок між уявленням, даними та логікою компонентів.
4. Використання типізованої мови Typescript запобігає множині помилок у процесі розробки.
5. Angular можна використовувати для створення мобільних та десктоп додатків.

Як правило, Angular додаток складається з наступних елементів:

1) компоненти;
2) послуги;
3) модулі.

Компоненти - це основні «будівельні блоки» інтерфейсу користувача. За своєю суттю компоненти є звичайним класом, до якого додано наступний декоратор:

```
@Component({ selector:
'селектор', templateUrl:
'шаблон'
})
```

Селектор позначає ім'я, яким даний компонент буде додаватися в інші, шаблон — шлях до html-файлу, відповідний даної компоненті.

Будь-який Angular додаток має головний модуль, що називається AppModule. Модуль є класом, позначеним декоратором @NgModule, функція якого приймає об'єкт з полями declarations (сюди входять компоненти, директиви, канали), exports (класи, які будуть використовуватися в інших модулях), imports (модулі, класи яких необхідні для використання в компонентах поточного модуля), providers (сервіси), bootstrap (компонент, що викликається під час завантаження програми).

Сервіси є класами, позначені декоратором @Service. Дані класи виконують завдання зберігання та отримання даних, представляють канал взаємодії між різними компонентами та інкапсулюють бізнес логіку.

SPA (single page application) — програма, яка є веб-сайтом або звичайною програмою, розміщеною на одній веб-сторінці.

Архітектура SPA програм влаштована таким чином, що при переході на нову сторінку оновлюється лише частина контенту. Таким чином, немає необхідності повторно завантажувати одні й самі елементи.

Основні переваги SPA додатків:

1) Висока швидкість роботи програми.

Оскільки SPA не оновлює всю сторінку, а лише необхідну частину, це значно підвищує швидкість роботи.

2) Висока швидкість розробки.

Готові бібліотеки пропонують потужні інструменти для розробки веб додатків. Над проектом можуть паралельно працювати back-end та front-end розробники. Завдяки чіткому поділу вони не заважатимуть один одному.

3) Мобільні додатки.

SPA дозволяє легко розробити мобільний додаток на основі готового коду.

Недоліки SPA додатків:

1. Погана пошукова оптимізація.

SPA працює на основі JavaScript та завантажує інформацію на запит з боку клієнта. Пошукові системи важко імітувати цю поведінку, тому більшість сторінок недоступні для сканування пошуковими системами.

2.2.2. Важливість мережі

Програма не працюватиме, якщо користувач відключить JavaScript у браузері.

Як СУБД використовується PostgreSQL.

PostgreSQL – об'єктно-реляційна СУБД з відкритим вихідним кодом.

Як об'єктно-реляційна СУБД PostgreSQL підтримує об'єкти користувача та їх поведінки, в тому числі операції, індекси, функції, типи даних. Дана

СУБД може працювати з числовим, текстовим, булевим типом, а також з грошовим, що перераховується, геометричним, xml, JSON типом.

Для гарантії цілісності даних PostgreSQL надає такі механізми як первинні та зовнішні ключі, унікальні та перевірочні обмеження, обмеження NOT NULL. Ця СУБД відповідає стандарту ANSI_SQL: 2008, відповідає вимогам ACID (Atomicity, Consistency, Isolation, Durability), тобто атомарності, узгодженості, ізольованості та надійності).

PostgreSQL придатна для зберігання більших обсягів інформації. Її показники представлені у таблиці 2.2.

Таблиця 2.2 - Показники PostgreSql.

Максимальний розмір БД	Не обмежений
Максимальний розмір таблиці	32 TB
Максимальний розмір рядка	1.6 TB
Максимальний розмір поля	1 GB
Максимальна кількість рядків в таблиці	Необмежено
Максимальна кількість стовпців в таблиці	250-1600 залежно від типу стовпця
Максимальна кількістьіндексів в таблиці	Необмежено

2.2.3. Відкриті системи та стандарти

На рисунку 2.2 представлено діаграму таблиць бази даних

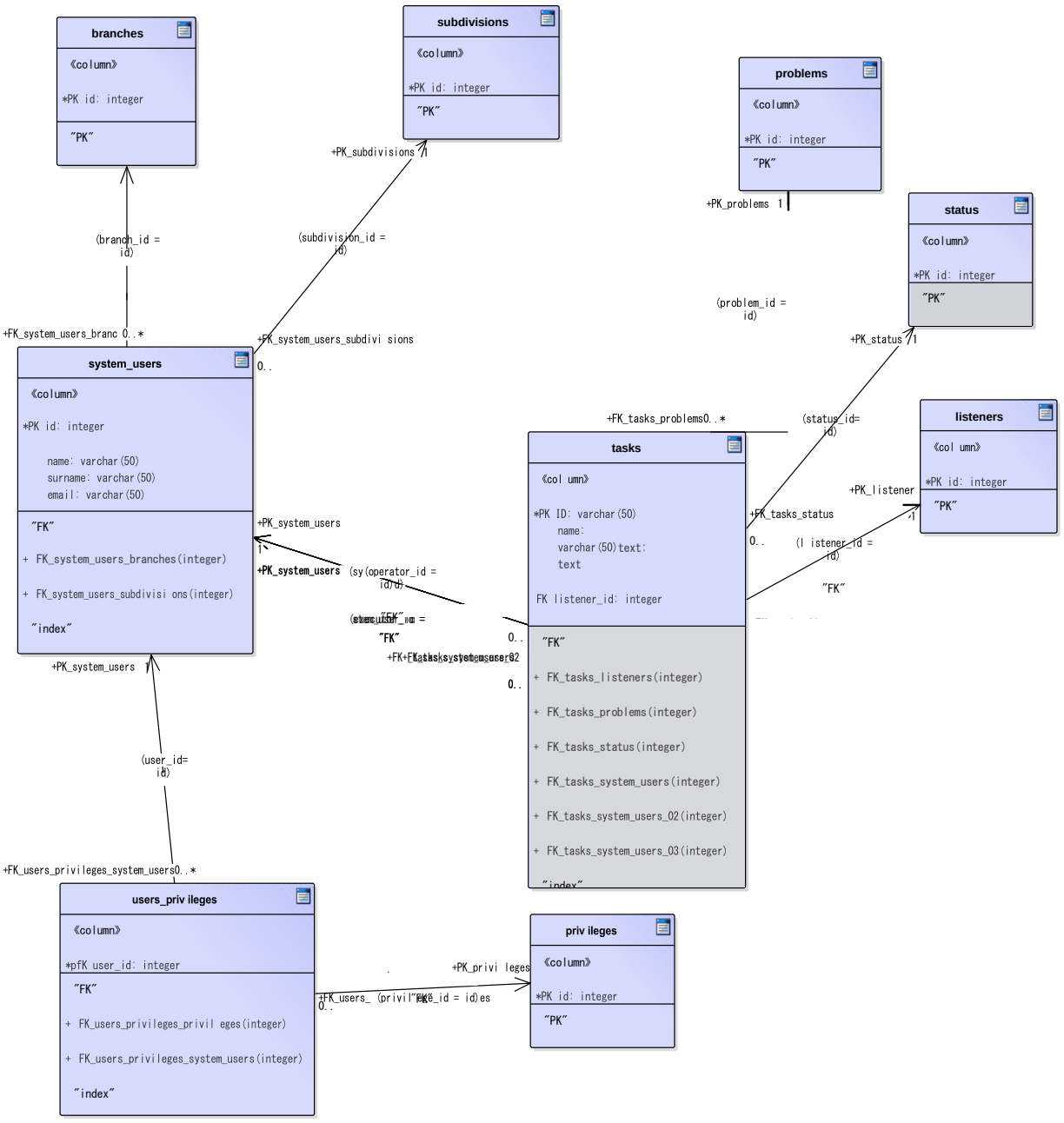


Рисунок 2.2-діаграма таблиць БД

У базі даних будуть такі таблиці:

1. Branches - містить інформацію про філії підприємства.
2. Subdivisions містить інформацію про відділи всередині підприємства.

3. Problems – містить типові проблеми.
4. Privileges – містить ролі користувачів.
5. Tasks - містить інформацію про заявки.
6. Status – містить інформацію про можливий статус заявок.
7. System_Users – містить користувачів системи.

2.3. Модель клієнт-сервер в Інтернеті

Spring Boot додаток створюється за допомогою Spring Initializer, який дозволяє вибрати компоненти, що використовуються в додатку, а потім зібрати проект.

Класи Spring Boot програми були розміщені за наступними пакетами:

1) config - для класів конфігурацій;
2) controllers - для класів-контролерів, що опрацьовують запити;
3) exceptions — для винятків користувача;
4) jwt - для класів, пов'язаних з автентифікацією;
5) mappers - для класів, які служать для перетворення об'єктів;
6) models - для jpa сутностей та репозиторіїв;
7) services - для сервісів.

Для роботи з Angular встановлюється сервер NodeJS та пакетний менеджер npm. Після цього встановлюється бібліотека Angular за допомогою команди npm install-g @angular/cli.

Для створення проекту з командного рядка використовується команда ng new
`ім'я проекту.`

Структура Angular програми має такий вигляд:

1) components – містить прості компоненти;
2) containers - містить складні компоненти, які є окремою сторінкою в

додатку;

3) services - містить класи сервісів;
4) store містить класи, які пов'язані зі сховищем станів NgRx;
5) guards-містить клас для обмеження доступу маршрутам користувачів, які не мають на це прав;
6) interceptors — класи, які перехоплюють та обробляють вхідні та вихідні запити.

3. ПРАКТИЧНА ЧАСТИНА

3.1. Клієнт-серверні обчислення

JWT (JSON Web Token) — JSON об'єкт, який визначається стандартом RFC 7519. JWT служить для безпечної передачі між двома учасниками.

JWT складається з 3 частин - заголовка (header), корисного навантаження (payload) та підпису (signature). Тобто JWT - це рядок у форматі header-base64-string. payload-base64-string. signature-base64-string.

Header є також JSON об'єкт. Він складається із двох полів
— typ, що описує тип токена, та alg, що визначає алгоритм шифрування. Таким чином, заголовок описує як обчислювати JWT підпис.

Корисне навантаження - це також JSON об'єкт, який стандартно містить три поля: sub -предмет, iss - додаток, що відправляє токен, exp
— час життя токена.

3.2. 3-рівнева архітектура

Ключові об'єкти контексту Spring Security:

1. SecurityContextHolder забезпечує доступ до SecurityContext.
2. SecurityContext містить Authentification та інформацію про запит у разі потреби.
3. Authentification представляє користувача (Principal), який включає в себе GrantedAuthority.
4. GrantedAuthorityвідображає дозволи для всьогододатку, надані користувачеві (Principal).
5. UserDetails містить необхідну інформацію для створення об'єкта

Authentification із DAO або іншого джерела даних безпеки.

6. UserDetailsService допомагає створювати UserDetails з імені користувача на основі рядка.
7. SecurityContext встановлюється викликом SecurityContextHolder.getContext(). SetAuthentication (…)
8. AuthenticationEntryPoint обробляє виняток AuthenticationException.

Коли http запит надходить на сервер, запит проходить через ланцюжок фільтрів. У цей ланцюжок фільтрів було додано JwtAuthTokenFilter.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
       http.cors().and().csrf().disable().authorizeRequests()
                   .antMatchers("/api/auth/**").permitAll()
                   .anyRequest().authenticated()
                   .and()
                   .exceptionHandling().authenticationEntryPoint(unauthorizedHand
ler).and()
                   .session
Management().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        http.addFilterBefore(authenticationJwtTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
 }
```

JwtAuthTokenFilter здійснює валідацію токена прі допомоги JwtProvider. Якщо литокен існує іваліден, то з нього витягується інформація про користувача та на основі неї створюється AuthenticationToken,який зберігається в Security Context, в іншому випадку Authentication Entry Point обробляє виняток Authentication Exception. @Override

```
 protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {
        try {
              String jwt = getJwt (request); if
              (jwt!=null &&
tokenProvider.validateJwtToken(jwt)) {
                   String username =
tokenProvider.getUserNameFromJwtToken(jwt);

                   UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
                   UsernamePasswordAuthenticationToken
authentication= new
Username Password Authentication  Token(userDetails, null,
userDetails.getAuthorities());
                   authentication. Set Details(new
```

```
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);
                  }
            } catch (Exception e) {
                  logger.error("Can NOT set user authentication
-> Message: {}", e);
            }

            filterChain.doFilter(request, response);
      }
```

Після цього кроку об'єкт AuthenticationToken передається в метод authenticate інтерфейсу AuthenticationManager.

Клас ProviderManager імплементує цей інтерфейс. ProviderManager спробує аутентифікувати користувача за допомогою набору AuthenticationProviders, потім поверне повністю заповнений об'єкт аутентифікації або згенерує виняток. Один з провайдерів — DaoAuthenticationProvider, який автентифікує користувача шляхом порівняння паролів з AuthenticationToken та UserDetailService. Конфігурація цього провайдера здійснюється в класі WebSecurityConfig.

```
@Override
     public void configure(AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {
           authenticationManagerBuilder
                     .userDetailsService(userDetailsService)
                     .passwordEncoder(passwordEncoder());
     }
```

Якщо автентифікація пройшла успішно, можна отримати UserDetails з об'єкта Authentication. У класі UserDetailService описаний метод, що дозволяє отримати об'єкт UserDetails за логіном користувача.

```
public UserDetails loadUserByUsername(String email) throws
                  UsernameNotFoundException {
           SystemUser user = userRepository.findByEmail(email)
                      .orElseThrow(() -> new
UsernameNotFoundException("User Not Found with -> username або
email : " + email)
           );

           return UserPrinciple.build(user);
```

```
}
```

Spring Security надає анотації @PreAuthorize, @PreFilter, @PostAuthorize та @PostFilter для перевірок авторизації до та після виклику методу та фільтрації.

Для дозволу використання подібних виразів використовується глобальна інструкція у класі WebSecurityConfig.

```
@EnableGlobalMethodSecurity(
        prePostEnabled = true
)
```

Тепер перед кожним методом контролера можна вказати роль користувача, який матиме право викликати цей метод.@PreAuthorize("hasRole('ADMIN')").

3.3. Минуле та майбутнє клієнтів

Auth guard використовується для запобігання доступу до маршрутів користувачів, які не мають на це прав. Auth guard містить єдиний метод canActivate(). Даний метод повертає true, якщо користувачеві дозволено доступ і false в іншому випадку, перед цим переправивши користувача на сторінку входу в систему.

```
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot)
{
const currentUser = this.authenticationService.currentUserValue;
if (localStorage.getItem('currentUser') && currentUser) { if (route.data.roles) {
let allowedRole = false; currentUser.user_details.forEach((e) => {
if (route.data.roles.includes(e['authority'])) { allowedRole = true;
}
});
if (allowedRole) {
return true;
}
this.router.navigate(['/auth/login']);return false;
}
return true;
}

this.authenticationService.logout();
this.router.navigate(['/auth/login']); return false;
}
```

Auth guard використовується при маршрутизації так.

```
{
path: 'admin',
component: AdminContainerComponent, Activate:
```

```
[AuthGuard],
data: {roles: [Role.Admin]},}
```

Error interceptor перехоплює всі відповіді сервера і перевіряє їх у наявність помилок. Якщо статус помилки 401, тобто це помилка аутентифікації, буде виконано вихід із системи і користувач буде перенаправлений на сторінку входу в систему.

```
intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
return next.handle(request).pipe(catchError((error, caught) => {
this.handleAuthError(error);return of
(error);
}) as any);
}
private handleAuthError(err: HttpErrorResponse): Observable<any> {
if (err.status === 401) { this.authenticationService.logout();
location.reload(true);
return of(err.message);
}
throw err;
}
```

JWT interceptor перехоплює всі запити до сервера і додає до них заголовок авторизації у вигляді рядка Bearer token, якщо користувач зареєстрований у системі.

```
intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
const token = JSON.parse(localStorage.getItem('currentUser')); if
(token) {
request = request.clone({
setHeaders: {
Authorization: `Bearer ${token.token}`
}
});
}
return next.handle(request);
}
```

Authentication Service використовується для реєстрації та аутентифікації користувача. Метод login відправляє запит на вхід до системи та отримує токен. Цей токен зберігається у локальне сховище. Це зроблено для того, щоб користувач міг залишатися в системі між сеансами браузера, докизакінчиться термінжиттятокенаабокористувач не вийде з системи.

Розшифроване значення токена заноситься в змінну поточного

```
User.login(body) {
```

```
return this.http.post<any>(`http://localhost:8090/api/auth/signi n`, body)
.pipe(map(data => {
if (data && data.token) { localStorage.setItem('currentUser',
JSON.stringify(data)); this.currentUserSubject.next(jwt_decode(data.token));
this.currentUser = this.currentUserSubject.asObservable();
}
return data;
}));
```

У методі hasRole перевіряється, чи наділений користувач необхідною роллю для виконання певного методу.

```
public hasRole(role: string) { let flag =
false;
if (this.currentUserSubject.value) { for (const
user of
this.currentUserSubject.value.user_details) {

if (user['authority'] ===
role) { flag = true;
}}}
return flag;
}
```

Залежно від ролі користувача можна приховувати або показувати певний вміст. Такий підхід реалізований для створення меню.

```
<li *ngIf="hasRole(role.User)"><a routerLink="/mytasks">
<div class="button">
<i class="material-icons" style="font-size: 30px"> work_outline
</i>
My tasks
</div>
</a>
</li>
```

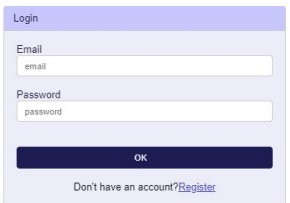


Рисунок 3.2-Форма входу до системи

Приклад валідації форми представлений рисунку 3.3.

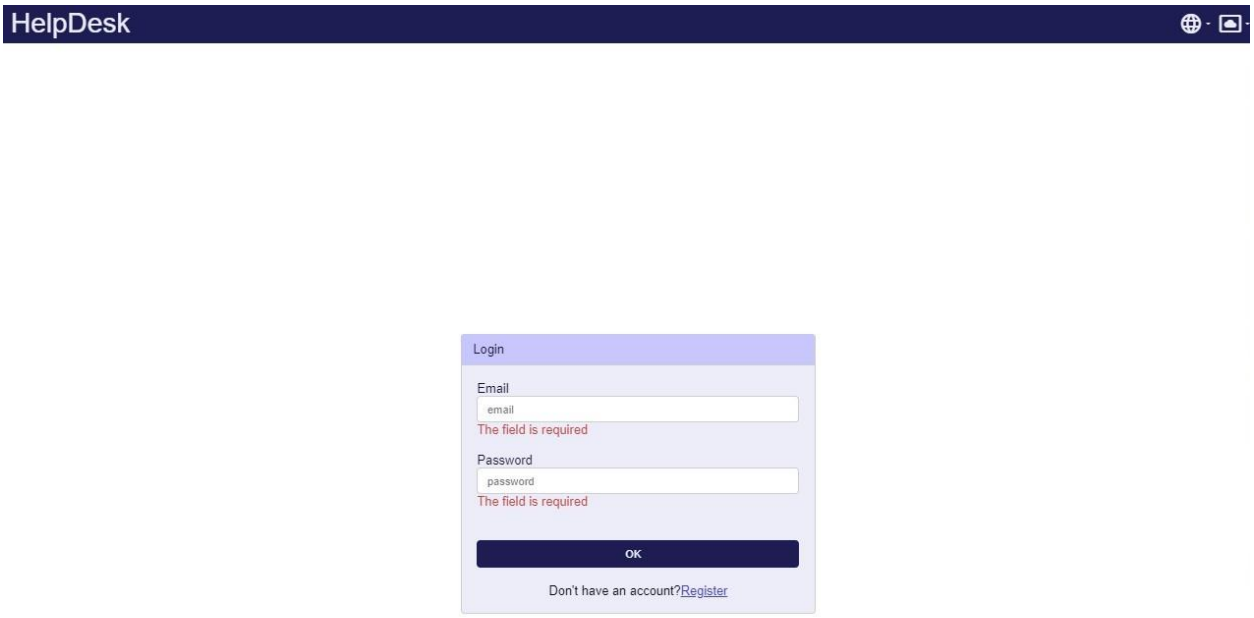


Рисунок 3.3-Приклад валідації форм

Сторінка поточних заявок містить форму для надсилання заявки та таблицю для відстеження змін за створеними заявками. При надсиланні заявки користувачеві необхідно ввести назву заявки, її опис та вибрати відділ, куди заявку буде надіслано. Заявку буде збережено в журналі заявок разом із датою її створення.

Форма відправки заявки:

```
         <input type="text" i18n-placeholder="@@title" placeholder="title"
formControlName="title"/>
         <div *ngIf="submitted && f.title.errors">
            <div *ngIf="f.title.errors.required">Title is required</div>
         </div></div>
      <div class="input-container">
         <label i18n="@@description">
            Description
         </label>
         <textarea placeholder="description" i18n-
placeholder="@@description"
formControlName="description"></textarea>
      </div>
      <div class="input-container">
         <label i18n="@@send-to-listener"> Send to
            listener
```

```
         </label>
         <select formControlName="listenerId">
            <option [value]="listener.id" *ngFor="let listener of
listeners">
               {{listener.subdivisionId['name']}}
            </option>
         </select>
         <div *ngIf="submitted && f.listenerId.errors">
            <div
*ngIf="f.listenerId.errors.required">Listener is required</div>
         </div> </div>
      <div class="button">
         <app-button-with-spinner [title]="'OK'"
            [showSpinner]="loading"
         >
         </app-button-with-spinner>
      </div>
<div *ngIf="error" i18n="@@error">Error</div>
   </div>
</form>
```

Користувач може видалити заявку, доки вона не призначена виконавцю. Як тільки заявку призначено виконавцю, кнопка видалення стає недоступною.

```
<button class="delete-task-button"
*ngIf="task.statusId.id === 1"
(click)="deleteTask(task.id)">
<i class="material-icons" style="font-size: 30px"> clear
</i>
</button>
```

Управління поточними заявками здійснюється за допомогою набору бібліотек NgRx. З цього набору бібліотек використовуються NgRx/store для зберігання стану програми та NgRx/effects для додавання даних до сховища. Завдяки використанню NgRx зменшується кількість запитів до бази; створюється окреме джерело даних - сховище; зміна стану робиться можливим лише з використанням спеціальних функцій, тобто стан не можна змінити безпосередньо; компоненти стають простішими.

Код ефекту для отримання поточних заявок наведено нижче.

```
@Effect()
   loadValue$: Observable<any> = this.actions$.pipe(
      ofType(taskActions.FETCH_ITEM),
      map((action: Fetch) => action.payload),
      switchMap((payload) => {
         return this.tasksService.getPersonalTasks(payload)
            .pipe(map((value) => {this.store.dispatch(new
AddTotal(value['totalPages']))});
          return new taskActions.AddAll(value['content']);
              }),
              catchError(error => of(new
taskActions.Error(error)))
            );
      })
   );
```

Сторінка поточних заявок представлена рисунку 3.4.

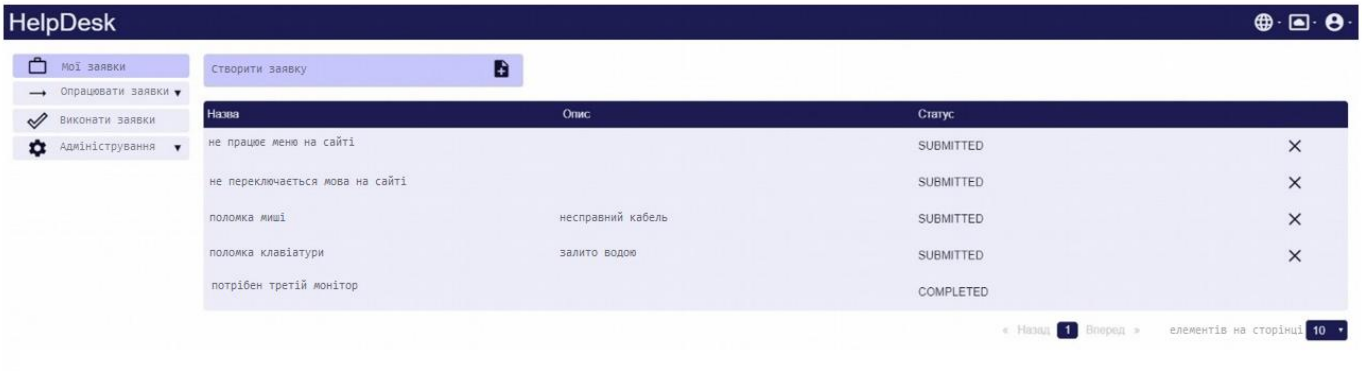


Рисунок 3.4- Сторінка поточних заявок

Сторінки оператора містять дві таблиці. Перша таблиця показує заявки, які потрібно розподілити між виконавцями. Для кожної заявки відображено її назву, опис та дату створення. Оператор обирає виконавця із запропонованого списку людей, що працюють у його відділі, і дату завершення заявки. Статус заявки після цього змінюється на Pending та заявка зникає з таблиці.

Друга таблиця містить виконані заявки, які потрібно закрити. Для цих заявок відображаються такі поля: назва, опис, дата створення, дата виконання, дата завершення. При закритті заявки оператором, заявка набуває статусу Closed.

Сторінки розподілу та закриття заявок представлені на рисунку 3.5.

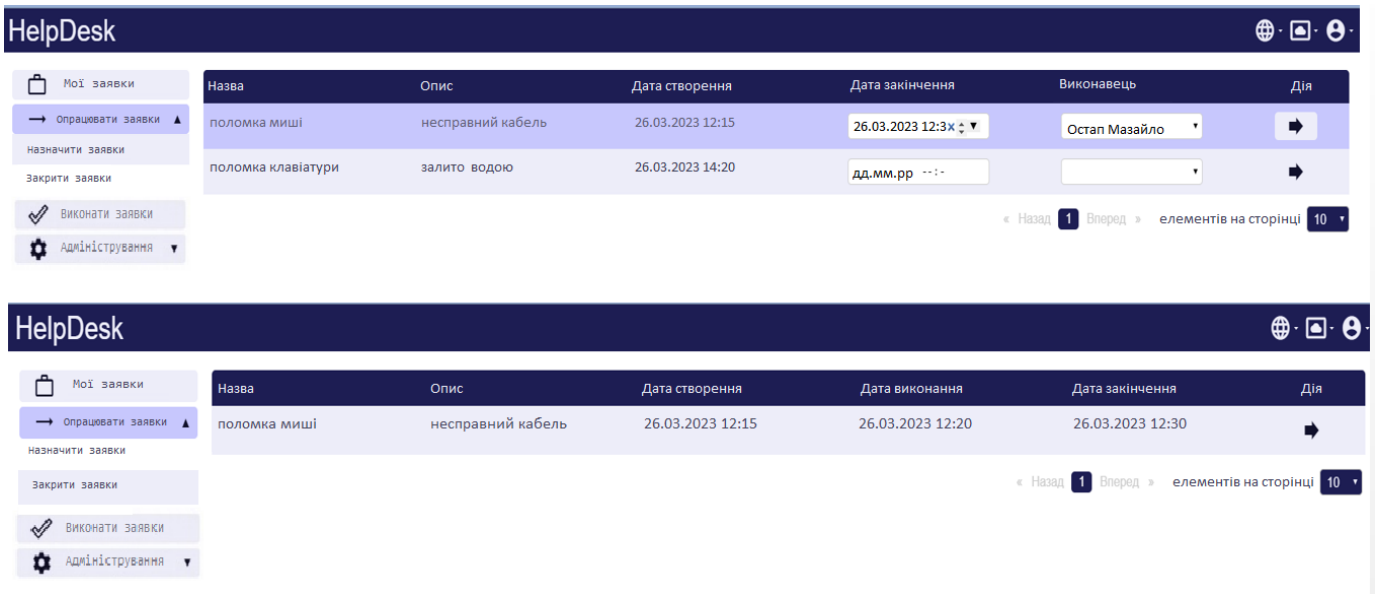
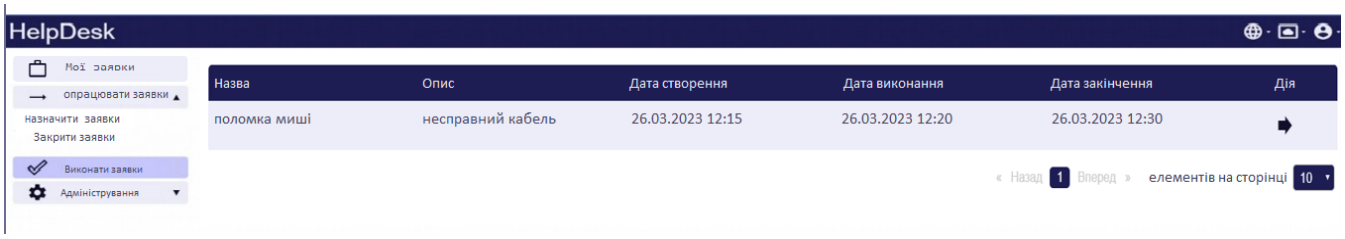


Рисунок 3.5-Сторінки розподілу та закриття заявок

Сторінка виконання заявок містить заявки, розподілені конкретному виконавцю. Для заявки відображено ім'я, опис, статус, дату створення, дату закінчення. Після того, як заявку буде виконано, її статус зміниться на Completed. Заявка залишиться в таблиці, доки її не закрито оператором.

Сторінку виконання заявок представлено рисунку 3.6.



Малюнок 3.6-Сторінка виконання заявок

На сторінці адміністрування реалізовано управління користувачами, наділення користувачів різними правами, управління підрозділами та відділами, а також створення звіту. Звіт містить дані з таблиць, вибраних користувачем, та зберігається у форматі xlsx.

Код створення звіту.

```
public exportAsExcelFile(sheets: Sheet[]): void { let sheetObj = {};
      sheets.forEach( e => { sheetObj[e.fileName] =
XLSX.utils.json_to_sheet(e.data);
      });
const sheetNames = sheets.map(({ fileName }) => fileName);
      const workbook: XLSX.WorkBook = { Sheets: sheetObj, SheetNames: sheetNames
};
      const excelBuffer: any = XLSX.write(workbook,
{ bookType: 'xlsx', type: 'array' });
   }
```

Сторінки управління користувачами та підрозділами представлені на рисунках 3.7-3.8.

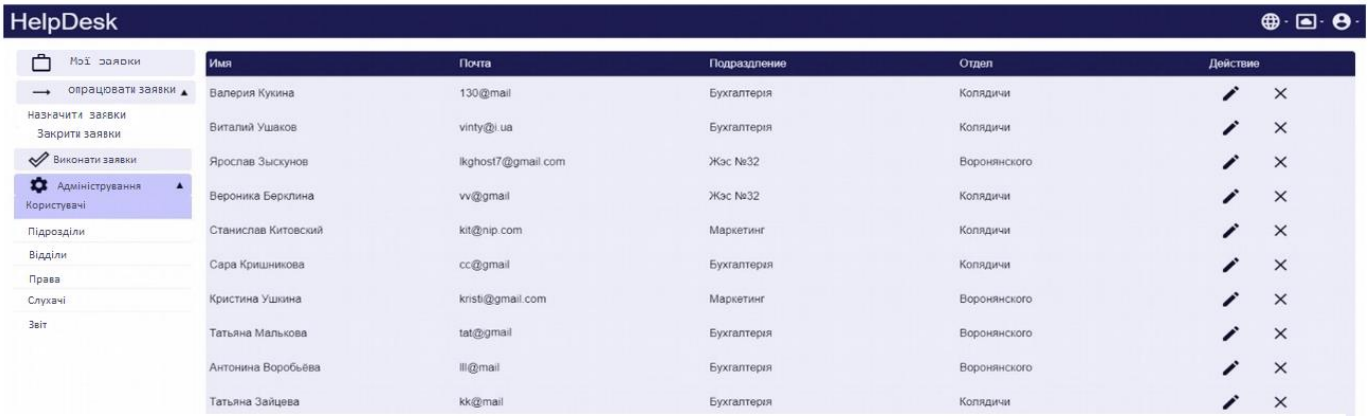


Рисунок 3.7-Сторінка адміністратора, управління користувачами

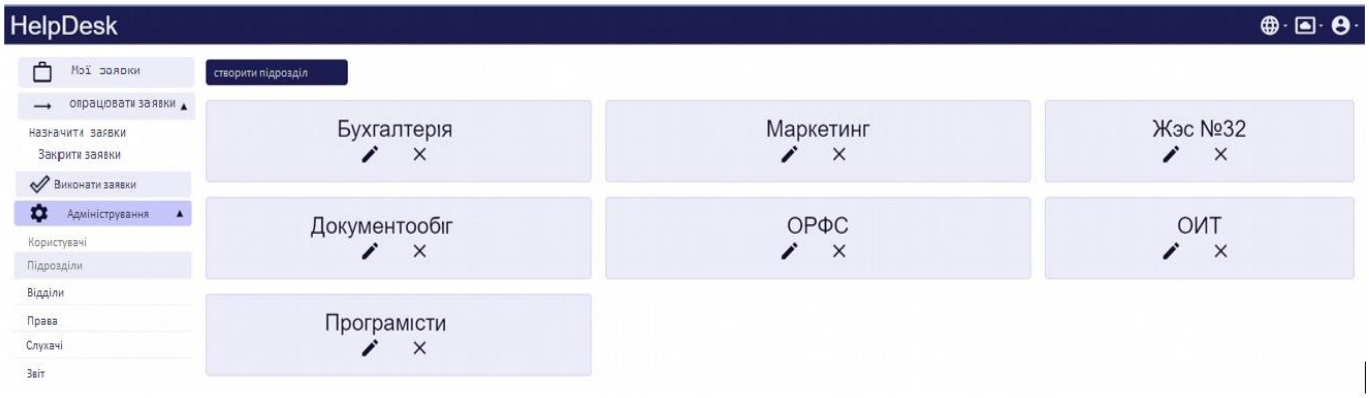


Рисунок 3.8-Сторінка адміністратора, управління підрозділами

Для всіх таблиць у додатку реалізовано пагінацію. Пагінація дозволяє розбити великий обсяг даних частини, звані сторінками. Кожна сторінка має окремий запит даних. У параметрах запиту вказується сторінка та кількість елементів на сторінці.

Розглянемо реалізацію пагінації для таблиці із поточними заявками. У серверній додатку для цього репозиторій успадковується від класу PagingAndSortingRepository з бібліотеки org.springframework.data.repository і метод пошуку завдань користувача передається об'єкт класу Pageable, який містить номер поточної сторінки і кількість елементів на сторінці. Метод поверне об'єкт класу Page, який міститиме масив елементів для поточної сторінки та кількість сторінок.

```
@Transactional@Rep
ository
public interface TasksRepository extends
PagingAndSortingRepository<Task, Long> {
      Page<Task> findAllBySystemUserId(SystemUser systemUser,
Pageable pageable);
}
```

Для пагінації клієнтської частини програми використовується бібліотека ngx-pagination, яка спрощує створення кнопок переходів між сторінками таблиці. Даний фрагмент представляє тіло таблиці з використанням каналу пагінації, який передається кількість елементів на сторінці, номер поточної сторінки і кількість сторінок.

```
<div class="table-row"
           *ngFor="let task of tasks | async | paginate:
{itemsPerPage: sizePerPage, currentPage: page, totalItems:
totalPages }">
      <div class="cell"><div class="cell-name"
i18n="@@title">Title</div><div>{{task.name}}</div></div>
      <div class="cell"><div class="cell-name"
i18n="@@description">Description</div><div>{{task.text}}<
/div></div>
      <div class="cell"><div class="cell-name"
i18n="@@status">Status</div><div>{{task.statusId.name}}</div></div>
      <div class="cell cell-small">
```

```
          <div class="cell-name" i18n="@@action">Action</div>
          <button class="delete-task-button"
   *ngIf="task.statusId.id === 1"
   (click)="deleteTask(task.id)">
   <i class="material-icons" style="font-size:
30px">clear</i></button></div></div>
```

Також визначено кнопки перемикання між сторінками та випадаючийсписок, що дозволяє вибрати, яку кількість елементів показувати на сторінці.

```
   <pagination-controls class="controls_paginator"
         previousLabel="Backwards" nextLabel="Forward"
         i18n-previousLabel="@@backwards" i18n-
         nextLabel="@@forward"
         (pageChange)="changePage($event)"
      ></pagination-controls>
      <div class="pagination-per-page">
         <div i18n="@@elements-per-page">Elements per page</div>
         <label>
            <select [(ngModel)]="sizePerPage" (change)="changeElements()"
   class="field-select">
               <option value="10">10</option>
               <option value="20">20</option>
               <option value="50">50</option>
               <option value="100">100</option>
            </select>
         </label>
      </div>
   </div>
```

Зміна теми програми ґрунтується на глобальних змінних css та використанні колірної моделі HSL. HSL – колірна модель, в якій колір задається трьома компонентами: тон, насиченість та світло. Тон визначається в межах від 0 до 360 градусів, насиченість та світла – від 0 до 100%. Чим більший показник насиченості, тим соковитішим вийде колір. Чим вище світла, тим більше буде яскравість кольору.

У головному файлі стилів було визначено змінні кольори.

```
   --red-hue: 360;
   --blue-hue: 240;
   --green-hue: 120;
   --main-hue: var(--blue-hue);
   --theme-darkness: 1;
   --error: hsl(var(--red-hue), 50%, 50%);
   --dark-bg-color: hsl(var(-main-hue), 50%, 20%);
   --midle-light-bg-color: hsl(var(-main-hue), 93%, 88%);
```

```
   --light-bg-color: hsl(var(-main-hue), 50%, 95%);
   --v: hsl(var(-main-hue), 50%, 10%);
   --midle-light-text-color: hsl(var(--main-hue), 50%,
50%);
   --light-text-color: hsl(var(-main-hue), 50%, 95%);
```

Перші три змінні відповідають за тон різних тем оформлення клієнтської програми. При зміні теми змінну main-hue поміщається вибраний тон. Потім слідують по три змінні на колір заднього фону і колір тексту, вони використовують main-hue як одну з трьох складових у колірній моделі HSL, решта двох складових варіюються для отримання більш темного або світлого кольору.

Метод зміни теми програми:

```
     changeTheme(color) {
                document.documentElement.style.setProperty('--main-hue', 'var(--' +
color + '-hue)');
     }
```

Безпосередньо у самих файлах стилів компонент ці кольори викликаються так:

```
color: var(--dark-text-color);
```

Дизайн програми є адаптивним. Це означає, що програма коректно відображається на різних пристроях: комп'ютерах, планшетах, смартфонах. Для використання різних стилів залежно від ширини пристроїв використовуються медіа-запити. Приклад медіа-запиту наведено нижче:

```
@media (max-width: 950px) .grid-container {
         grid-template-columns: auto auto;
    }
}
```

Даний медіа-запрос спрацьовує, коли ширина вікна або екрана менше 950 пікселів, при цьому змінюється властивість grid-template-columns у css-класі grid-container.

Приклад відображення сторінки поточних заявок на смартфоні наведено на рисунку 3.9.

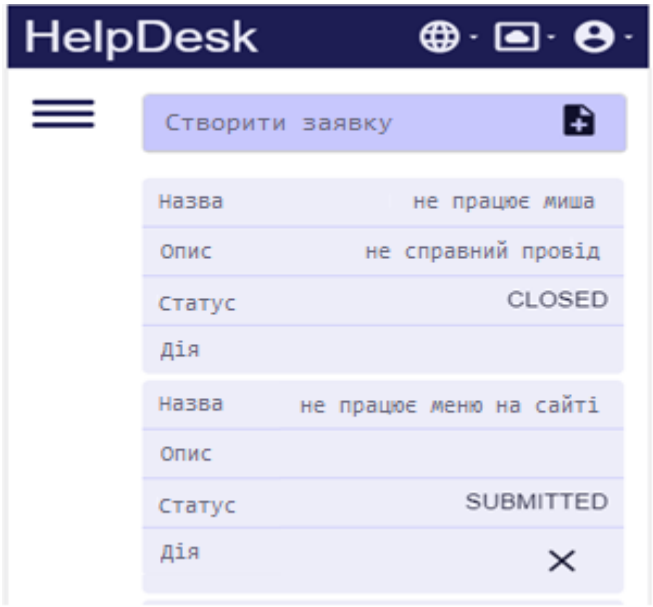


Рисунок 3.9 — Мобільна версія

Для здійснення локалізації було створено 3 файли: locale.xlf, locale.en.xlf, locale.ru.xlf. Останні два файли містять переклади рядків англійською та українською мовами відповідно. Для кожного перекладу у файлах визначено блок такого вигляду:

```
<trans-unit id="users" datatype="html">
         <source>Users</source>
         <target>Користувачі</target>
</trans-unit>
```

У цьому блоці id є унікальним ідентифікатором перекладу, source є рядком у шаблоні, який буде заміщений на переклад із поля target.

У файлах шаблонів додається атрибут i18n тег блоку, який необхідно перекласти.

```
<div class="cell-name" i18n="@@title">Title</div>
```

Дати, числа та грошові одиниці відображатимуться по-різному залежно

від обраної мови, таким чином здійснено інтернаціоналізацію.

Для клієнтської програми використовується AoT (Ahead-of-Time) компіляція. Це означає, що код на мові Typescript та Angular HTML буде конвертований у JavaScript та HTML на етапі складання, а не виконання.

Компіляція під час процесу збирання допомагає забезпечити швидшу візуалізацію в браузері, заздалегідь виявити помилки та знизити можливість виникнення ін'єкційних атак.

Код налаштування україномовної версії програми представлений нижче:

```
  "
  "baseHref": "/ua/",
   "i18nFile": "src/locale/messages.ua.xlf", "i18nFormat": "xlf",
   "i18nLocale": "ua", "i18nMissingTranslation": "error"
},
  production-ua": {
                        "optimization": true, "outputHashing": "all",
                        "outputPath": "dist/browser/ua/", "sourceMap":
                        false,
                        "extractCss": true,
                        "namedChunks": false, "aot":
                        true, "extractLicenses": true,
                        "vendorChunk": false,
                        "buildOptimizer": true,
                        "fileReplacements": [
                           {
                              "replace": "src/environments/environment.ts",
                              "with": "src/environments/environment.prod.ts"
                           },
                           {
                              "replace": "src/locale/locale-en.ts", "with":
                              "src/locale/locale-ua.ts"
                           }
                        ],
```

4. БЕЗПЕКА ЖИТТЕДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Аварії з викидом радіоактивних речовин

За даними світової статистники, що фіксується в головній службі небезпечних ситуацій, великі аварії на підприємствах та об’єктах різних типів, де лінійні розміри зон ураження досягають декілька сотень або навіть тисяч метрів, є досить рідкісними. Проте тим не менш, у світі в середньому за рік відбувається близько 2 – 3 подібних аварій.

Аварії із загибелью понад 25 осіб і числом поранених більше 100 реєструється в середньому раз на 2,5 роки.

У цілому, як вважають фахівці, спостерігається неухильне зростання кількості пропислових і енергетичних аварій, викликане, з одного боку, збільшенням кількості небезпечних об’єктів, з іншого боку, зростанням питомої щільності населення в зонах розвитку промислових і енергетичних об’єктів.

Найбільша в історії людстава радіаційна катастрофа на Чорнобильській АЕС сталась 26 квітня у 1986 році. Кілька років після катастрофи всі офіційни джерела СРСР повідомляли, що жертвами Чорнобиля стали тільки 33 людини – в основному пожежники, які брали участь в наймерших роботах. Потім почали з’являтись окремі повідомлення про те, що від променевої хвороби загинуло кілька десятків ліквідаторі, а захворіли тисячі. Про жертви серед місцевого населення не говорилось взагалі. Режим секретності з питань аварай ЧАЕС, який існував до 1991 року, не дозволяв відтворити об’єктивну картину масштабів ураження населення.

За сучасними уявленнями, аварія на ЧАЕС має серйозні наслідки пролонгованої дії, в тому числі такі, що можуть виявлятись на генетичному рівні в окремих груп персоналу АЕС, ліквідаторів і населення, яке проживає поблизу зони аварії.

У результаті вибуху четвертого реактора Чорнобильської атомної електростанції стався величезний викид радіоактивних речовин в атмосферу. Ці радіоактивні опади випали в основному в межах євро-азіатського континенту.

За оцінками, протягом 1986–1987 рр. до ліквідації наслідків аварії було залучено понад 350000 чоловік-«ліквідаторів» з числа військовослужбовців, працівників АЕС, місцевої міліції та пожежних служб. Досить високі дози радіації отримали близько 240000 чоловік під час проведення робіт з ліквідації наслідків аварії в межах 30-кілометрової зони, виконання робіт з консервації аварійного 4-го блоку АЕС – будівництва «Саркофагу», очищення дахів АЕС, створення системи захисту водних об'єктів.

У даний час приблизно п'ять мільйонів людей проживають в районах Білорусі, Російської Федерації та України, де рівні радіоактивного забруднення грунтів цезієм перевищують 37 кБк/м2. З них приблизно 270000 людей продовжують жити в районах, які класифікувалися повноважними органами як зони посиленого контролю ,де зараження 137Cs перевищує 555 кБк/м2.

Можна припустити збільшення кількості випадків смерті від раку протягом усього життя серед осіб, які зазнали впливу радіації в результаті аварії. У зв'язку з тим, що в даний час неможливо визначити, які конкретні випадки раку були викликані радіацією, кількість таких випадків смерті можна оцінити лише статистично на основі використання інформації та проекцій, отриманих під час досліджень на людях, які вижили після вибухів атомних бомб [20, с. 202, 210]

4.2 Долікарська допомога при обмороженні

Гіпертермія – це патологічний стан організму, виникає внаслідок значного порушення терморегуляції або дії зовнішнього тепла.

Замерзання – це загальне патологічне переохолодження організму, зумовлене поступовим зниженням температури тіла під впливом охолодження дії зовнішнього середовища, при недостатній захисній терморегуляторній функції організму. В основі замерзання лежить порушення терморегуляції організму. Загальна гіпотермія викликає зниження усіх видів обміну, в результаті чого створюються умови, за яких тепловіддача значно перевищує теплоутворення.

Залежно від глибини ураження тканини розрізняються на чотири ступеня відмороження (рис. 4.1):

– Перший ступінь – шкіра постраждалого блідого кольору, незначно набрякла, чутливість знижена або повністю відсутня. Підвищується чутливість до холоду, що може зберігатись 2-3 місяці і більше;

– Другий ступінь – у ділянці відмороження утворюється міхури, наповнені прозорою або білою рідиною. До цього характерні підвищення температури тіла, остуда;

– Третій ступінь – омертвіння шкіри: з’являються міхурі, наповнені рідиною темно-червоного або темно-бурого кольору як результат обмороження глибокої дерми. Характерний розвиток інтоксикації – остуда, потовиділення, значне погіршення самопочуття, апатія;

– Четвертий ступінь – поява міхурів, наповнених чорною рідиною. У постраждалого є ознаки шоку. Розвиток набряку відбувається через 1 або 2 год. Набряк, як правило, поширюється на проксимальні відділи кінцівок. Потім розвивається муміфікація, рідше – волога гангрена.

Рисунок 4.1 – Стадії обмороження

Дії при наданні першої медичної допомоги відрізняються в залежності від ступеню обмороження, стану загального охолодження організму людини, його віку та наявних хвороб.

При обмороженні постраждалого в першу чергу потрібно перенести у тепле приміщення, після чого приступити до поступового зігрівання. Застосовують пасивне зігрівання: знімаючи мокрий та весь холодний одяг, поступово обсушуючи шкіру, накривання теплими ковдрами. Цей метод ефективний при легкій гіпотермії. Обкладання постраждалого грілками або поступове занурення у теплу воду призведе до руху відносно холодної крові по всьому тілу з подальшим охолодженням і порушень у життєвоважливих органах, тому краще обійтись без цього.

При гіпотермії тяжкого ступеня, коли температура тіла нижче 30 градусів, потрібно застосувати активне зовнішнє зігрівання, що включає в себе: ковдри з підігрівом, гарячі ванни.

Розмерзання, розігрівання тканин та відновлення кровообігу повинно поширюватись у зворотньому напрямку (від центра до периферії) під дією тепла власного тіла та крові. Передчасне розігрівання тканини на периферії без відновлення кровообігу веде до їх загибелі.

Тому для ушкоджених холодом тканин потрібно створити умови термоса, шляхом накладання термоізоляційних пов’язок від кисті до плечових суглобів та ступні докульшових суглобів, обгорнувши їх поліетиленовою плівкою, поверх плівки накласти товстий шар вати чи шерстяних тканин і добре забинтувати марлевими бинтами. При цьому забезпечивши постраждалому часте пиття теплої води та якнайшвидшу госпіталізацію до найближчого лікувального закладу [6, с. 10, 24].

ВИСНОВКИ

У ході дипломної роботи було спроектовано та розроблено Help Desk систему з урахуванням напрямків програми розвитку інформатизації України. Основними завданнями даної системи є: управління інцидентами, проблемами, запитами обслуговування, здійснення комунікацій з користувачами. Управління інцидентами забезпечує відновлення послуги найшвидшим способом та допомагає знизити вплив інцидентів на бізнес, а управління проблемами допомагає усунути повторне виникнення інцидентів, скоротити витрати на їх вирішення, підвищити доступність послуг та продуктивність персоналу. Завдяки цьому забезпечується безперервність та безвідмовність інформаційних потоків,

З цією системою можуть працювати спеціалісти IT-відділів у галузі реагування на проблеми надання IT-послуг; адміністративно-господарського відділу з питань матеріально-технічного забезпечення; відділу документаційного забезпечення оформлення договірної роботи.

У результаті дипломного дослідження вирішено такі завдання:

1. Здійснено аналіз предметної галузі з погляду проблеми обслуговування заявок та засобів розробки необхідного програмного забезпечення. Порівняли існуючі системи Happydesk, Freshdesk, OTSR, Spiceworks.
2. Розглянуто стандарти у сфері програмного забезпечення. А саме такі: "Системи менеджменту якості", "Інформаційні технології", "Системна та програмна інженерія", "Програмна інженерія". З їхньою допомогою дано тлумачення основних понять.
3. Створено діаграми потоків даних, а також IDEF0, UML діаграми. Дані діаграми є порядок дій обслуговування заявок, логічну схему бази даних.

4. Розроблено веб-додаток, який дозволяє підтримувати процеси обслуговування заявок у межах обраної предметної області. Серверний додаток реалізований із застосуванням Spring Boot, клієнтський - бібліотеки Angular.

ПЕРЕЛІК ПОСИЛАНЬ

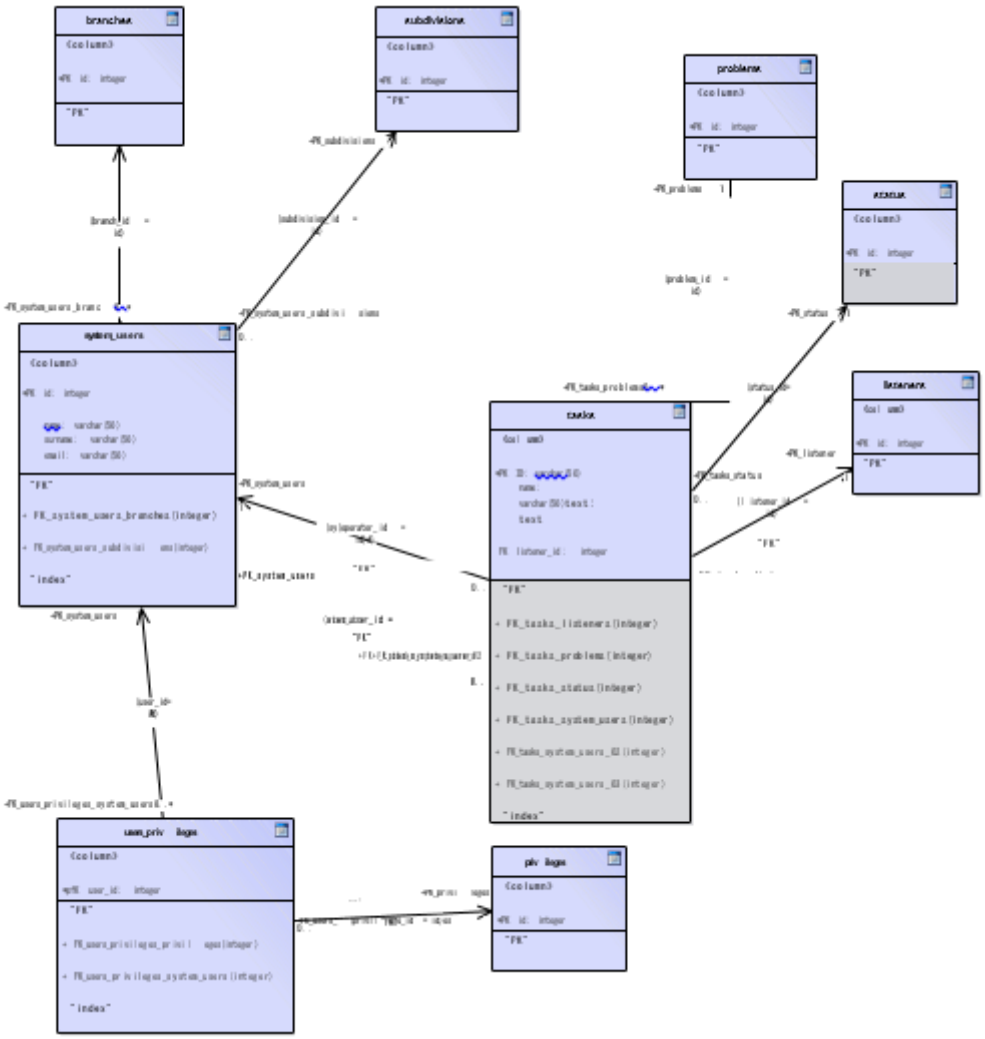
1. Державна програма розвитку цифрової економіки та інформаційного суспільства на 2016 – 2020 роки. Постанова Кабінета Міністрів України у 23.03.2022 № 235.
2. Кошуняєва, Н.В. Теорія масового обслуговування/Н.В. Кошуньова, Н.М. Патронова. - Архангельськ: САФУ, 2013. - 107 с.
3. Стратегія розвитку інформатизації в Україні на 2016 - 2022 роки. Затверджено на засіданні Президії Ради Міністрів від 03.11.2015 № 26.
4. СУБД [Електронний ресурс] Режим доступу:https://habr.com/post/348220/ - Дата доступу: 01.12.2022.
5. Freshdesk Help Desk [Електронний ресурс]. - Режим доступу: https://freshdesk.en/ - Дата доступу: 01.12.2022.
6. Happydesk Help Desk [Електронний ресурс]. - Режим доступу: https://happydesk.en/ - Дата доступу: 01.12.2022.
7. Information technology - Service management - Part 10: Concepts and vocabulary: ISO/IEC 20000-10:2022(en). - Technical Committee ISO/IEC JTC 1, 2022. - 28p.
8. OTRS Help Desk [Електронний ресурс]. - Режим доступу: https://doc.otrs.com/doc/manual/admin/4.0/ua/html/otrs.html - Дата доступу: 01.12.2022.
9. PostgreSQL [Електронний ресурс]. - Режим доступу: https://habr.com/ua/post/282764/. - Дата доступу: 17.03.2022.
10. Quality management systems - Fundamentals and vocabulary: ISO 9000:2015(en). - Technical Committee ISO/TC 176, 2015 - 51p.
11. Ramachandran, S.Four Waysto Improve Digital Customer Service / S. Ramachandran, H. Jenkac, N. Clark, P. Despontin. [Електронний ресурс]. - Режим доступу: https://www.bcg.com/publications/2017/four-ways-improve-digital-customer-service.asp. -дата доступу:01.03.2022
12. Ross, DT Structured Analysis (SA): A Language for Communicating Ideas / Ross DT - Waltham: IEEE Transactions on software engineering, 1977. - 19 p.
13. Software Engineering - Software Life Cycle Processes - Maintenance: ISO/ IEC 14764:2006(en). - Technical Committee ISO/IEC JTC 1/SC, 2022.- 44p.
14. SPA програми [Електронний ресурс]. - Режим доступу:http://merehead.com/blog-ru/single-page-application-vs-multi-

page-application. - Дата доступу: 17.03.2022.
15. Spiceworks Help Desk [Електронний ресурс]. - Режим доступу: https://www.spiceworks.com/free-help-desk-software/online/ - Дата доступу: 01.12.2022.
16. Spring Boot [Електронний ресурс]. - Режим доступу: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm - Дата доступу: 17.03.2022.
17. Systems and software engineering - Life cycle management - Part 1: Guidelines for life cycle management: ISO/IEC/IEEE 24748-1:2018(en).- Board approval 27.09.2022. - Technical Committee ISO/IEC JTC 1/SC, 2022. -72p.
18. Systems and software engineering — Requirements for acquirers and suppliers of information for users: ISO/IEC/IEEE 26512:2018(en). - Publication date :2022-06. - Technical Committee ISO/IEC JTC 1/SC 7, 2022. -37p.
19. Systems and software engineering — Software life cycle processes: ISO/IEC 12207:2008(en). - Technical Committee ISO/IEC JTC 1/SC, 2008. - 122p.
20. Плачкова С.Г., Плачков І.В. Енергетика: історія, сучастність і майбутнє – м. Київ 2013 р. с. 301– ISBN 978-966-8163-18-0
21. В.О. Крилюк, Домедична допомога Медичний посібник – 2014. – 84 с.ISSN 1681-2751

ДОДАТКИ

Додаток А

діаграма таблиць бази даних



Додаток Б
Лістинг код форми заявки

```
         <input type="text" i18n-placeholder="@@title" placeholder="title"
formControlName="title"/>
         <div *ngIf="submitted && f.title.errors">
            <div *ngIf="f.title.errors.required">Title is required</div>
         </div></div>
      <div class="input-container">
         <label i18n="@@description">
            Description
         </label>
         <textarea placeholder="description" i18n-
placeholder="@@description"
formControlName="description"></textarea>
      </div>
      <div class="input-container">
         <label i18n="@@send-to-listener"> Send to
            listene
         </label>
         <select formControlName="listenerId">
            <option [value]="listener.id" *ngFor="let listener of
listeners">
               {{listener.subdivisionId['name']}}
            </option>
         </select>
         <div *ngIf="submitted && f.listenerId.errors">
            <div
*ngIf="f.listenerId.errors.required">Listener is required</div>
         </div> </div>
      <div class="button">
         <app-button-with-spinner [title]="'OK'"
            [showSpinner]="loading"
         >
         </app-button-with-spinner>
      </div>
<div *ngIf="error" i18n="@@error">Error</div>
   </div>
</form>
```