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

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка системи менеджменту та моніторингу проєктів на серверах з використанням технологій Python, Django

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

	(підпис)	Лань О. П. (прізвище та ініціали)
Керівник	(підпис)	Мудрик І. Я. (прізвище та ініціали)
Нормоконтроль	(підпис)	(прізвище та ініціали)
Завідувач кафедри	(підпис)	(прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2024

РЕФЕРАТ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2024, Сторінок 51, таблиць 1, рисунків 16, презентація.

Тема: Розробка системи менеджменту та моніторингу проєктів на серверах з використанням технологій Python, Django [1].

У даній дипломній роботі розглядається процес розробки системи менеджменту та моніторингу проєктів на серверах з використанням сучасних технологій програмування Python та Django. Метою роботи є створення ефективного інструменту для управління та контролю за проєктами, що розгорнуті на серверних платформах, забезпечення надійності, безпеки та зручності в роботі з даними.

У роботі представлено детальний огляд архітектури системи, описано етапи розробки, включаючи проектування бази даних, реалізацію серверної частини, створення інтерфейсу користувача, а також інтеграцію різних модулів для забезпечення моніторингу та аналізу стану проєктів. Особлива увага приділена питанням безпеки даних та оптимізації продуктивності системи.

Результатом роботи є повнофункціональний веб-додаток, який дозволяє менеджерам проєктів та адміністраторам серверів ефективно керувати проєктами, відслідковувати їхній стан у режимі реального часу та вчасно реагувати на можливі проблеми. Система надає інтуїтивно зрозумілий інтерфейс для управління користувачами, налаштувань сповіщень та звітів про діяльність проєктів.

Ключові слова: система менеджменту проєктів, моніторинг серверів, Python, Django, веб-додаток, управління даними, безпека, продуктивність.

ANNOTATION

Bachelor's qualification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2024, Pages 51, tables 1, figures 16, presentation.

Topic: Development of a project management and monitoring system on servers using Python and Django technologies

This thesis explores the development of a project management and monitoring system on servers using modern programming technologies, Python and Django. The aim of the work is to create an efficient tool for managing and monitoring projects deployed on server platforms, ensuring reliability, security, and ease of data handling.

The thesis provides a detailed overview of the system architecture, describing the stages of development, including database design, server-side implementation, user interface creation, and integration of various modules to ensure monitoring and analysis of project status. Special attention is given to data security and system performance optimization.

The result of the work is a fully functional web application that allows project managers and server administrators to efficiently manage projects, monitor their status in real-time, and promptly respond to potential issues. The system provides an intuitive interface for managing users, notification settings, and project activity reports.

Keywords: project management system, server monitoring, Python, Django, web application, data management, security, performance.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у розвитку бізнесу, забезпеченні конкурентоспроможності та ефективності управління проєктами. Зі зростанням складності та масштабів проєктів виникає потреба в інструментах, які здатні забезпечити надійний менеджмент та моніторинг усіх аспектів реалізації проєктів. Ефективне управління проєктами, особливо тими, що розгорнуті на серверних платформах, потребує використання спеціалізованих систем, здатних контролювати різноманітні процеси, аналізувати дані в режимі реального часу та швидко реагувати на будь-які проблеми.

Метою цієї дипломної роботи є розробка системи менеджменту та моніторингу проєктів на серверах з використанням технологій Python та Django. Python є потужною та гнучкою мовою програмування, що дозволяє швидко та ефективно розробляти складні додатки. Django, у свою чергу, є одним із найпопулярніших веб-фреймворків для Python, який забезпечує зручний та швидкий спосіб створення веб-додатків, включаючи інструменти для роботи з базами даних, аутентифікації користувачів, маршрутизації URL та багато іншого.

Сучасний бізнес все частіше стикається із необхідністю управляти численними проєктами одночасно, забезпечуючи при цьому їхню своєчасну реалізацію, відповідність бюджетам та високі стандарти якості. Традиційні підходи до менеджменту проєктів часто не в змозі забезпечити необхідну гнучкість та прозорість, що робить актуальною розробку нових, більш ефективних систем. Використання веб-технологій для створення таких систем дозволяє забезпечити доступ до необхідної інформації з будь-якої точки світу та у будь-який час.

У рамках цієї роботи розглянуто процес розробки системи з точки зору проектування архітектури, реалізації серверної частини, створення інтерфейсу користувача та інтеграції модулів для моніторингу проєктів у реальному часі. Особлива увага приділена питанням безпеки даних, оптимізації продуктивності системи та забезпеченню зручності її використання для кінцевих користувачів.

Детально описано, як використання Python та Django дозволяє створювати масштабовані, надійні та легкі в обслуговуванні системи.

Крім того, робота охоплює аналіз існуючих рішень на ринку систем менеджменту та моніторингу проєктів, їхні переваги та недоліки, а також визначення вимог до нової системи. Розроблена система забезпечуватиме менеджерам проєктів та адміністраторам серверів ефективний інструмент для управління проєктами, моніторингу їх стану у реальному часі та своєчасного виявлення і вирішення проблем. Система також надасть можливості для управління користувачами, налаштування сповіщень, створення звітів про діяльність проєктів та аналізу даних для прийняття обґрунтованих управлінських рішень.

Даний проєкт розроблено спільно з Бизовим Іваном Сергійовичем, аспірантом, спеціальності 122 Комп'ютерні науки, Харківського національного університет імені В. Н. Каразіна. Тут продемонстровано лише частину, яка виконана мною особисто.

Ця дипломна робота має на меті не лише продемонструвати технічні аспекти створення такої системи, але й показати її практичне застосування у реальних умовах, підкреслюючи важливість сучасних технологій у покращенні процесів управління та моніторингу проєктів. Окрім того, у роботі будуть представлені перспективи подальшого розвитку та вдосконалення системи, зокрема шляхом інтеграції з іншими сучасними технологіями та інструментами для забезпечення ще вищої ефективності та функціональності.

Сучасні тенденції в області програмного забезпечення і методологій управління проєктами, таких як Agile і DevOps, підкреслюють необхідність інтеграції інструментів автоматизації та моніторингу. Це дозволяє командам розробників і менеджерам проєктів не лише спостерігати за поточним станом проєктів, але й прогнозувати потенційні проблеми та здійснювати дії для їх запобігання. В цьому контексті, розроблена система сприятиме підвищенню ефективності командної роботи, зниженню ризиків та покращенню якості кінцевого продукту.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

Із подібних систем можна виділити декількох конкурентів. Один із них – панель керування VDS.

Панель керування віртуального виділеного сервера (VDS) є інтерфейсом, який дозволяє користувачам керувати своїм віртуальним сервером та його ресурсами. Вона зазвичай включає різні інструменти та функції для адміністрування серверів, управління налаштуваннями, моніторингу продуктивності та інших завдань.

Рис. 1.1 – Вигляд сервісу VDS

Панель керування віртуального виділеного сервера (VDS) зазвичай включає кілька ключових розділів, які забезпечують повний контроль над сервером і його ресурсами. Інформаційна панель, або Dashboard, відображає загальну інформацію про сервер, включаючи його статус, IP-адреси, операційну систему та апаратні ресурси, такі як процесор, пам'ять і диск. Вона також показує останні дії, журнали подій та сповіщення, що дозволяє користувачам швидко оцінити стан сервера.

Розділ управління ресурсами дозволяє контролювати використання процесора, оперативної пам'яті та дискового простору. Користувачі можуть налаштовувати обмеження та пріоритети для процесорних ресурсів, відстежувати використання оперативної пам'яті та керувати її розподілом, а також управляти розділами диску та файлами.

Мережевий розділ забезпечує управління мережевими інтерфейсами та IP-адресами, налаштування брандмауера та мережевих правил, а також моніторинг трафіку та мережевої активності. Це дозволяє забезпечити безпеку та ефективність мережевих з'єднань.

Розділ операційної системи включає встановлення та оновлення операційної системи, доступ до консолі сервера для виконання команд та налаштувань, а також управління службами та процесами, які працюють на сервері. Це забезпечує стабільну роботу сервера та можливість швидкого реагування на проблеми.

Резервне копіювання та відновлення є критично важливим аспектом управління сервером. Цей розділ дозволяє створювати резервні копії даних та налаштувань сервера, налаштовувати автоматичне резервне копіювання та швидко відновлювати дані у випадку збоїв.

Моніторинг сервера дозволяє відстежувати його продуктивність у реальному часі, включаючи показники використання ресурсів, такі як CPU, RAM, диск та мережа. Сповіщення про перевантаження або збої у роботі сервера дозволяють оперативно вирішувати проблеми.

Безпека є одним з найважливіших аспектів управління сервером. Панель керування VDS дозволяє налаштовувати брандмауер та правила доступу, управляти користувачами та їхніми правами доступу, а також встановлювати та управляти SSL-сертифікатами для забезпечення безпечних з'єднань.

Додаткові інструменти та налаштування включають управління доменами та DNS-записами, налаштування автоматизації завдань (cron jobs), а також інтеграцію з іншими сервісами та додатками, що забезпечує гнучкість і розширює можливості управління сервером.

Панель керування VDS надає користувачам можливість ефективно керувати своїми віртуальними серверами завдяки різноманітним функціям. Управління користувачами та правами доступу дозволяє додавати, редагувати та видаляти користувачів, а також налаштовувати рівні доступу та права для забезпечення безпеки. Віртуальна консоль доступу забезпечує можливість виконання команд та

налаштувань безпосередньо на сервері, що дозволяє швидко реагувати на проблеми.

Автоматизація завдань, таких як резервне копіювання та оновлення системи, економить час та зусилля, забезпечуючи безперервність роботи сервера. Моніторинг і сповіщення дозволяють відстежувати продуктивність сервера та отримувати сповіщення у випадку проблем або перевантажень, що забезпечує своєчасне реагування на можливі загрози.

Панель керування VDS є потужним інструментом, який дозволяє користувачам ефективно управляти своїми віртуальними серверами, забезпечуючи їх стабільну та безперебійну роботу. Це включає зручність використання, моніторинг у реальному часі, автоматизацію завдань, підвищену безпеку, резервне копіювання та відновлення, гнучкість і масштабованість, а також інтеграцію з іншими сервісами.

Схожим сервісом є Hetzner control panel.

Панель керування Hetzner, також відома як Hetzner Robot, є веб-інтерфейсом, який надає користувачам зручний спосіб керування серверами від Hetzner та їхніми іншими хостинг-послугами. Ця панель керування дозволяє користувачам виконувати широкий спектр завдань, пов'язаних з адмініструванням серверів, управлінням ресурсами та налаштуванням мережевих параметрів.

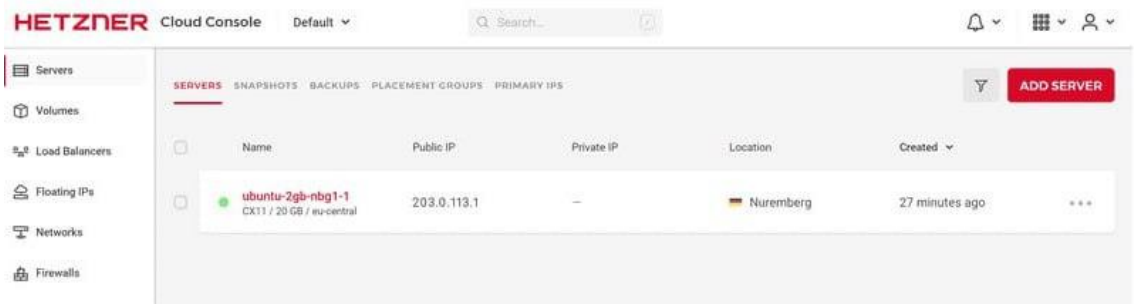


Рис. 1.2 – Вигляд панелі Hetzner

Основні функції панелі керування Hetzner включають управління серверами, моніторинг та статистику, управління мережевими налаштуваннями, резервне копіювання та відновлення, управління DNS, безпеку, а також білінг та оплату. Користувачі можуть переглядати інформацію про свої виділені сервери та

віртуальні приватні сервери (VPS), дистанційно вмикати та вимикати сервери, а також отримувати доступ до консолей серверів безпосередньо через веб-інтерфейс. Панель забезпечує моніторинг ресурсів у реальному часі, таких як процесор, пам'ять і дисковий простір, разом з історичними даними про продуктивність та сповіщеннями про проблеми та перевантаження.

Для управління мережевими налаштуваннями панель дозволяє налаштовувати IP-адреси, включаючи IPv4 та IPv6, керувати брандмауером та правилами доступу, а також налаштовувати VLAN та інші мережеві параметри. Функціонал резервного копіювання та відновлення включає створення резервних копій даних та налаштувань серверів, налаштування автоматичного резервного копіювання та відновлення даних з резервних копій у випадку збоїв. Функції управління DNS дозволяють користувачам створювати та керувати DNS-записами для своїх доменів, налаштовувати зони DNS та переглядати записи.

Безпека є критичним аспектом панелі керування Hetzner, яка забезпечує управління користувачами та правами доступу, двофакторну аутентифікацію для підвищення безпеки облікових записів, а також журнали активності користувачів та подій. Секція білінгу та оплати дозволяє користувачам переглядати рахунки та історію оплат, керувати платіжними методами та налаштовувати автоматичні оплати та сповіщення про наближення термінів оплати.

Панель керування Hetzner є потужним та зручним інструментом для управління серверами та іншими хостинг-послугами від Hetzner. Вона пропонує широкий спектр функцій для адміністрування, моніторингу та налаштування, а також підвищену безпеку та автоматизацію завдань. Хоча для новачків може знадобитися час на освоєння панелі, її можливості роблять її ефективним рішенням для управління інфраструктурою.

1.2 Порівняльний аналіз конкурентів

Для того щоб у правильному напрямку розвивати свій проект, необхідно обов’язково проаналізувати переваги та недоліки конкурентів та в подальшому на основі цих даних сформувати вимоги, що допоможуть розробити систему яка зможе зрівнятись із конкурентами або навіть перевершити їх.

Для панелі керування VDS можна виділити такі позитивні сторони:

- Зручність використання: інтуїтивно зрозумілий інтерфейс дозволяє легко налаштовувати та керувати сервером без необхідності глибоких технічних знань, доступ до основних функцій та налаштувань відбувається через єдиний інтерфейс.
- Моніторинг у реальному часі: можливість відстежувати стан сервера та використання ресурсів у реальному часі; отримання сповіщень про проблеми або перевантаження.
- Автоматизація завдань: налаштування автоматичних завдань, таких як резервне копіювання та оновлення системи, а також економія часу та зусиль на виконання рутинних завдань.
- Безпека: містить інтегровані інструменти для управління безпекою, такі як брандмауер, управління користувачами та SSL-сертифікати, можливість налаштування прав доступу для різних користувачів.
- Резервне копіювання та відновлення: легке створення резервних копій даних та налаштувань, швидке відновлення даних у випадку збоїв.
- Гнучкість та масштабованість: можливість легко збільшувати ресурси сервера відповідно до потреб. налаштування серверів під конкретні вимоги проєктів.
- Інтеграція з іншими сервісами: можливість інтеграції з популярними інструментами та сервісами для розробки та управління проєктами.

Мінуси панелі керування VDS:

- Вартість: деякі панелі керування можуть бути платними або вимагати додаткових ліцензій також можуть виникати додаткові витрати на підтримку та оновлення.
- Продуктивність: використання панелі керування може впливати на продуктивність сервера через споживання ресурсів, неправильне налаштування може призвести до зниження продуктивності.
- Обмеженість налаштувань: деякі панелі керування можуть не надавати доступ до всіх можливостей та налаштувань сервера, користувачі з глибокими технічними знаннями можуть виявити обмеження у налаштуваннях.
- Складність інтеграції: інтеграція з деякими специфічними інструментами або сервісами може бути складною або неможливою, існує потреба у додаткових налаштуваннях для забезпечення сумісності.
- Залежність від постачальника: використання наданих панелей керування може призвести до залежності від постачальника послуг, можливі труднощі при міграції на іншу панель або платформу.
- Технічна підтримка: якість технічної підтримки може варіюватися залежно від постачальника послуг, можливі затримки у вирішенні технічних проблем.

Панель керування VDS має багато переваг, таких як зручність використання, автоматизація завдань та підвищена безпека. Однак, існують і недоліки, включаючи можливі витрати, вплив на продуктивність та залежність від постачальника. Вибір панелі керування залежить від конкретних потреб користувача та ресурсів, які він готовий витратити на її підтримку.

Нижче наведені переваги Hetzner Control Panel

- Зручність у використанні: інтуїтивно зрозумілий інтерфейс дозволяє легко керувати серверами та ресурсами, виконання багатьох завдань проходить лише за кілька кліків.

- Широкі можливості налаштування: висока гнучкість у налаштуванні мережі, безпеки та управління ресурсами, адаптація серверів відповідно до конкретних потреб.
- Автоматизація завдань: автоматичне резервне копіювання, моніторинг продуктивності, що економить час і зусилля.
- Безпека: вбудовані інструменти безпеки, такі як двофакторна аутентифікація, детальне управління правами доступу.

Недоліки Hetzner Control Panel

- Крива навчання: для новачків панель може здатися складною через велику кількість функцій і налаштувань, вона потребує певного часу для повного освоєння.
- Обмежені можливості кастомізації: обмежена гнучкість у налаштуваннях інтерфейсу та функціоналу. Може бути незручним для користувачів зі специфічними вимогами.
- Залежність від інтернет-з'єднання: для доступу до панелі необхідне стабільне інтернет-з'єднання, у випадку проблем з інтернетом користувачі можуть втратити доступ до своїх серверів.

Hetzner Control Panel є потужним і зручним інструментом для управління серверами та хостинг-послугами, надаючи широкий спектр функцій для адміністрування, моніторингу та налаштування. Незважаючи на те, що для новачків може знадобитися час на освоєння панелі, її можливості та функціонал роблять її ефективним рішенням для управління інфраструктурою. Основні переваги включають зручність використання, широкі можливості налаштування, автоматизацію завдань і високий рівень безпеки. Однак деякі недоліки, такі як крута крива навчання для новачків, обмежені можливості кастомізації та залежність від інтернет-з'єднання, можуть впливати на користувацький досвід.

Отже, проаналізувавши переваги а недоліки вищезгаданих конкурентів можна зробити висновок, що дані сервіси надають великий спектр функцій, однак вони працюють лише для серверів у межах своєї компанії, а також не надають можливості зберігати команди для подальшого використання.

1.3 Обґрунтування вибору напрямку дослідження

Вибір напрямку дослідження у сфері веб-розробки програм на основі веб-сайтів обумовлений низкою факторів, таких як попит на ринку праці, доступність технологій, зручність та ефективність використання інструментів, а також можливості для кар'єрного росту. Веб-розробка є однією з найбільш затребуваних галузей в IT, оскільки сучасний бізнес, організації та приватні особи все більше залежать від онлайн-платформ для взаємодії зі своїми клієнтами та аудиторією.

Веб-розробка включає створення та підтримку веб-додатків, веб-сайтів та веб-сервісів, які дозволяють користувачам взаємодіяти з ними через інтернет-браузери. Цей процес включає роботу з різними технологіями, мовами програмування, фреймворками та інструментами для створення функціональних, естетичних та безпечних веб-рішень.

Основні компоненти веб-розробки включають фронтенд, бекенд та бази даних. Фронтенд включає HTML (мова розмітки для створення структури веб-сторінок), CSS (мова стилів для оформлення та візуального представлення веб-сторінок) та JavaScript (мова програмування для додавання інтерактивності та динамічного контенту на веб-сторінках). Крім того, широко використовуються фреймворки та бібліотеки, такі як React.js, Vue.js та Angular, які спрощують розробку складних фронтенд-додатків.

Бекенд включає серверні мови програмування, такі як Python, Ruby, PHP, Java та Node.js, які обробляють запити від клієнтів та взаємодіють з базами даних. Фреймворки, як-от Django, Flask, Ruby on Rails та Express.js, допомагають структурувати бекенд-код та надають безліч готових рішень для реалізації стандартних задач. Бази даних включають реляційні СУБД, такі як MySQL, PostgreSQL, SQLite, що використовують SQL для управління даними, та NoSQL бази даних, такі як MongoDB та CouchDB, які забезпечують гнучкість у зберіганні даних, що не відповідають традиційній реляційній моделі.

Інфраструктура та хостинг включають хмарні сервіси, як-от AWS (Amazon Web Services), Google Cloud Platform та Microsoft Azure, які забезпечують

масштабованість, надійність та безпеку веб-додатків. Сервери та контейнери, такі як Docker та Kubernetes, використовуються для управління та розгортання додатків.

Для реалізації цього проекту було обрано мову програмування Python та фреймворк Django. Python є однією з найбільш популярних та універсальних мов програмування, яка відзначається простотою синтаксису та високою читабельністю коду. Це робить її ідеальною для швидкої розробки та прототипування. Python широко використовується у веб-розробці, аналізі даних, машинному навчанні, автоматизації та багатьох інших сферах. Його велика та активна спільнота забезпечує багатий вибір бібліотек та інструментів, що значно полегшує процес розробки. Ця мова програмування надзвичайно проста у використанні та ідеально підходить для розробки веб-додатків.

Django є високорівневим веб-фреймворком для мови програмування Python, який забезпечує швидкий розвиток та зручну структуру для веб-додатків. Він дає дуже великий функціонал одразу, тобто вручну не все потрібно налаштовувати і сам фреймворк надає великі можливості. Він був обраний з наступних причин:

- Швидкість розробки. Django надає безліч готових до використання компонентів, таких як аутентифікація користувачів, управління сесіями, адмін-панель, що значно скорочує час на розробку та дозволяє зосередитися на бізнес-логіці додатка. Вбудована ORM (Object-Relational Mapping) дозволяє працювати з базами даних на високому рівні абстракції, що також прискорює процес розробки.
- Безпека. Django включає в себе засоби захисту від поширених веб-загроз, таких як SQL-ін'єкції, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Clickjacking. Вбудовані механізми для управління користувачами та правами доступу дозволяють легко забезпечити безпеку додатка.
- Гнучкість та масштабованість. Django підходить як для малих проектів, так і для великих високонавантажених систем. Він дозволяє легко масштабувати додаток, додавати нові функції та інтегруватися з іншими

системами. Великий вибір сторонніх пакетів та розширень дозволяє швидко додавати нові функціональні можливості.

- Документація та спільнота. Django має одну з найкращих документацій серед веб-фреймворків, що дозволяє легко навчатися та вирішувати проблеми під час розробки.

Отже, вибір Python та Django для цього проекту обумовлений їхньою популярністю, зручністю у використанні, високим рівнем безпеки та потужними можливостями для швидкої та ефективної розробки веб-додатків.

Для фронтенд-частини сайту вирішено не використовувати жодних фреймворків, оскільки сам фронтенд є невеликим і досить шаблонізованим, тому не було потреби витрачати значні ресурси на його розробку. Для створення інтерфейсу були використані HTML5, JavaScript, jQuery, Bootstrap та SCSS.

HTML5 – це остання версія мови розмітки HTML, яка використовується для структурування веб-сторінок. Вона включає нові елементи та атрибути для мультимедіа, графіки, форм і семантичної розмітки, що спрощує розробку та покращує доступність контенту.

JavaScript – це мова програмування, яка використовується для створення динамічних і інтерактивних елементів на веб-сторінках. Вона дозволяє маніпулювати HTML і CSS, обробляти події та взаємодіяти з веб-серверами.

jQuery – це популярна бібліотека JavaScript, яка спрощує написання коду для роботи з HTML-документами, обробки подій, анімації та взаємодії з сервером через AJAX. Вона дозволяє скоротити кількість коду та підвищити його читабельність.

Bootstrap – це фреймворк для розробки адаптивних веб-додатків. Він включає готові компоненти та стилі для створення форм, кнопок, навігаційних панелей та інших елементів інтерфейсу. Bootstrap значно прискорює розробку і забезпечує консистентний вигляд сайту на різних пристроях.

SCSS (Sassy CSS) – це препроцесор CSS, який розширює можливості стандартного CSS, додаючи змінні, вкладені правила, функції та інші корисні інструменти. SCSS допомагає організувати та підтримувати стилі, роблячи їх більш модульними та легкими для управління.

1.4 Вибір методології розробки

Методології розробки програмного забезпечення є системами принципів, практик і процедур, які керують процесом створення програмних продуктів. Вони визначають структуру роботи, ролі учасників, етапи розробки та методи контролю якості. Основною метою використання методологій є підвищення ефективності розробки, зниження ризиків і забезпечення відповідності продукту вимогам замовника.

Існує багато різних методологій, кожна з яких має свої переваги та недоліки. Класичні методології, такі як водоспадна модель, передбачають послідовне виконання етапів розробки, починаючи з аналізу вимог і завершуючи тестуванням та підтримкою. Цей підхід забезпечує чітку структуру процесу, але може бути недостатньо гнучким у випадку змін вимог.

З іншого боку, гнучкі методології, такі як Agile, орієнтовані на постійну адаптацію до змін та тісну співпрацю з клієнтом. Agile передбачає ітеративний підхід, при якому проект розбивається на короткі цикли, що дозволяє регулярно отримувати зворотний зв'язок та вносити корективи на ранніх етапах. Це сприяє швидшому реагуванню на зміни та підвищує якість кінцевого продукту.

Вибір методології залежить від багатьох факторів, таких як розмір та складність проекту, вимоги клієнта, команда розробників та ресурси. Кожна методологія має свої особливості, які впливають на процес розробки та кінцевий результат. Розуміння різних методологій та їх правильне застосування дозволяє оптимізувати процес розробки та створити програмне забезпечення, що відповідає очікуванням користувачів та бізнес-цілям.

Тому зважаючи на всі аспекти вибір припав на методологію Agile.

Agile є однією з найпопулярніших методологій розробки програмного забезпечення, відомою своєю гнучкістю, швидкістю реагування на зміни та орієнтацією на співпрацю з клієнтом. Вибір Agile для цього проекту обумовлений кількома ключовими факторами, які роблять цю методологію ідеальною для сучасних веб-проектів.

Agile базується на Маніфесті Agile, який визначає чотири основні цінності: люди та взаємодія важливіші за процеси та інструменти, робоче програмне забезпечення важливіше за всеосяжну документацію, співпраця з клієнтом важливіша за обговорення умов контракту, а готовність до змін важливіша за дотримання плану. Успіх проекту залежить від ефективної співпраці команди, а не від суворого дотримання процесів. Основний акцент робиться на створенні працюючого продукту, а не на створенні великої кількості документації. Постійний діалог з клієнтом дозволяє точніше визначити вимоги та швидше реагувати на зміни, а гнучкість і здатність швидко адаптуватися до змін є ключовими факторами успіху.

Основні принципи Agile включають ітеративний підхід, коли проект розбивається на короткі цикли (ітерації або спринти), кожен з яких триває зазвичай від одного до чотирьох тижнів. По завершенню кожного спринту команда представляє працюючий інкремент продукту. Після кожного спринту команда проводить ретроспективу, аналізуючи успіхи та помилки для покращення процесу розробки. Тісна взаємодія з клієнтом дозволяє отримувати зворотний зв’язок та коригувати вимоги на ранніх етапах. Команди мають автономію та відповідальність за організацію своєї роботи, що сприяє підвищенню мотивації та продуктивності. Постійне тестування та інтеграція забезпечують високу якість продукту на кожному етапі розробки.

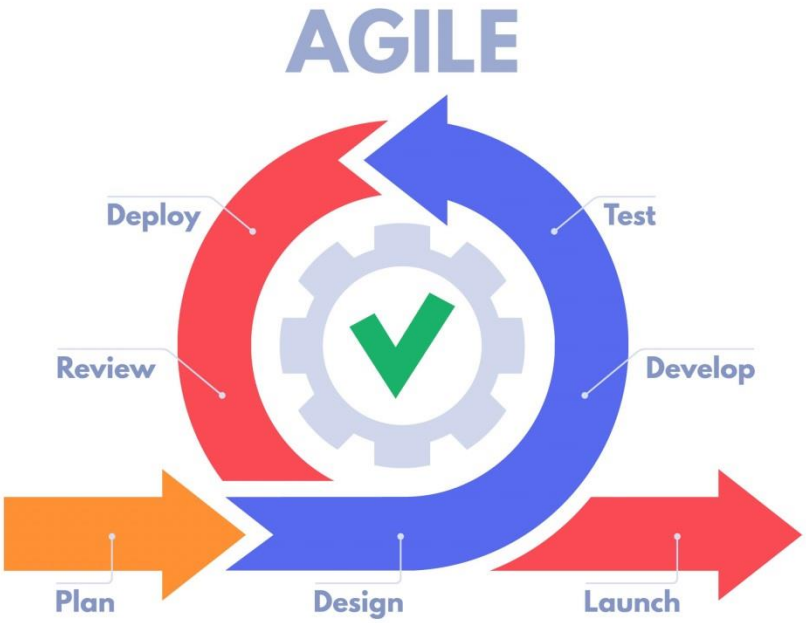


Рис. 1.3 – Цикл методології Agile

Для реалізації проекту з розробки системи менеджменту та моніторингу проєктів на серверах з використанням технологій Python та Django, Agile підходить ідеально через кілька причин. Гнучкість у розробці дозволяє команді швидко адаптуватися до змін у вимогах клієнта чи умов ринку, забезпечуючи швидке реагування на зворотний зв'язок та нові виклики. Постійна взаємодія з клієнтом та всередині команди дозволяє швидко виявляти проблеми та приймати ефективні рішення, що підвищує загальну якість продукту. Ітеративний підхід забезпечує регулярне постачання функціонального програмного забезпечення, що дозволяє клієнту бачити результати на кожному етапі та вносити корективи, якщо це необхідно. Agile дозволяє краще зрозуміти та задовольнити потреби кінцевих користувачів через постійний зворотний зв'язок та тестування.

Вибір Agile як методології розробки для цього проекту дозволяє забезпечити високу якість програмного забезпечення, швидко реагувати на зміни, ефективно співпрацювати з клієнтом та зосередитись на потребах користувачів. Цей підхід сприяє створенню продукту, який відповідає вимогам ринку та очікуванням клієнтів, забезпечуючи успіх проекту.

1.5 Формування вимог

Вимоги до системи є фундаментальним аспектом процесу розробки програмного забезпечення, що визначає, як система повинна функціонувати та взаємодіяти з користувачами і зовнішніми системами. Вимоги можна розділити на дві основні категорії: функціональні та нефункціональні.

Функціональні вимоги описують конкретні функції та завдання, які система повинна виконувати. Вони включають у себе операції, які система повинна підтримувати, умови введення і виведення даних, а також інші поведінкові аспекти. Прикладом функціональних вимог можуть бути: можливість створення користувачів, управління базами даних, генерація звітів або взаємодія з іншими програмними системами.

Нефункціональні вимоги визначають атрибути системи, які не стосуються конкретних функцій, але є важливими для загальної якості та ефективності системи. Вони охоплюють аспекти продуктивності, безпеки, масштабованості, надійності, зручності користування та сумісності. Наприклад, нефункціональні вимоги можуть включати вимоги до швидкості відгуку системи, рівня безпеки даних, підтримки великої кількості користувачів одночасно або забезпечення доступності системи протягом 99,9% часу.

Збалансоване врахування функціональних і нефункціональних вимог забезпечує створення ефективного, надійного і зручного програмного забезпечення, яке відповідає потребам користувачів і бізнес-цілям.

Функціональні вимоги до системи:

- Керування серверами: можливість додавання, видалення та редагування налаштувань серверів, віддалений доступ до серверів через SSH або інші протоколи, перезавантаження та вимкнення серверів.
- Моніторинг ресурсів: відображення поточного стану ресурсів (CPU, RAM, дисковий простір, мережевий трафік), налаштування та перегляд сповіщень про перевищення порогових значень ресурсів.

- Управління користувачами: створення, редагування та видалення облікових записів користувачів, ризначення ролей та прав доступу для різних користувачів.
- Безпека: аутентифікація та авторизація користувачів, відстежування дій користувачів (логування), підтримка двофакторної аутентифікації (2FA).
- Резервне копіювання: налаштування автоматичних резервних копій, відновлення даних з резервних копій.
- Управління мережею: налаштування та управління IP-адресами, конфігурація брандмауера та інших мережевих налаштувань.
- Інтеграція з іншими сервісами: API для інтеграції з зовнішніми системами, підтримка веб-хуків для повідомлень про події.

Нефункціональні вимоги:

- Продуктивність: система повинна забезпечувати високу швидкість відгуку на дії користувачів, має бути підтримка одночасної роботи великої кількості користувачів без зниження продуктивності.
- Надійність: система повинна бути доступною 99.9% часу, автоматичне відновлення після збоїв.
- Масштабованість: система повинна підтримувати легке додавання нових серверів та користувачів без необхідності суттєвих змін у конфігурації.
- Безпека: захист від атак типу DDoS, SQL-ін'єкцій, XSS, шифрування даних при передачі та зберіганні.
- Зручність користування: інтуїтивно зрозумілий інтерфейс, докладна документація та довідкові матеріали для користувачів.
- Сумісність: підтримка основних веб-браузерів, сумісність з різними операційними системами серверів.
- Обслуговування: легкість оновлення та розширення функціональності системи, можливість моніторингу та аналізу логів для виявлення та усунення проблем.

2 РОЗРОБКА СИСТЕМНОЇ АРХІТЕКТУРИ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка системної архітектури та програмного забезпечення є одним з ключових етапів у створенні сучасних інформаційних систем. Цей процес включає визначення структурних компонентів системи, їх взаємодію та функціональні можливості, а також розробку програмного забезпечення, яке реалізує ці можливості. Правильне проектування архітектури є критичним для забезпечення ефективності, масштабованості, надійності та зручності використання кінцевої системи.

На етапі дизайну архітектури системи визначаються основні компоненти, їх взаємозв'язки та взаємодії, а також обираються технології та інструменти, що будуть використані для реалізації. Це дозволяє створити чітку структуру, яка буде основою для подальшої розробки програмного забезпечення. Правильний дизайн архітектури допомагає уникнути багатьох проблем на етапах розробки, тестування та впровадження, забезпечуючи гнучкість і можливість адаптації до змін вимог.

Розробка програмного забезпечення включає процес створення коду, який реалізує визначені функціональні можливості та відповідає встановленим вимогам до якості. Цей процес включає проектування інтерфейсів користувача, написання програмного коду, тестування та інтеграцію окремих компонентів у цілісну систему. Застосування передових методологій та інструментів розробки дозволяє забезпечити високу якість коду, знизити кількість помилок та підвищити продуктивність роботи.

Успішна реалізація проектів у сфері розробки системної архітектури та програмного забезпечення потребує глибоких знань, досвіду та вміння працювати в команді. Важливо не тільки дотримуватись технічних стандартів та методологій, але й враховувати потреби кінцевих користувачів, забезпечуючи створення зручних, ефективних та надійних рішень.

2.1 Вибір архітектури проєкту

Для створення даного проєкту було обрано монолітну архітектуру.

Моноліт — це архітектурне рішення, у якому усі компоненти та модулі тісно пов’язані між собою, та залежать один від одного [4].

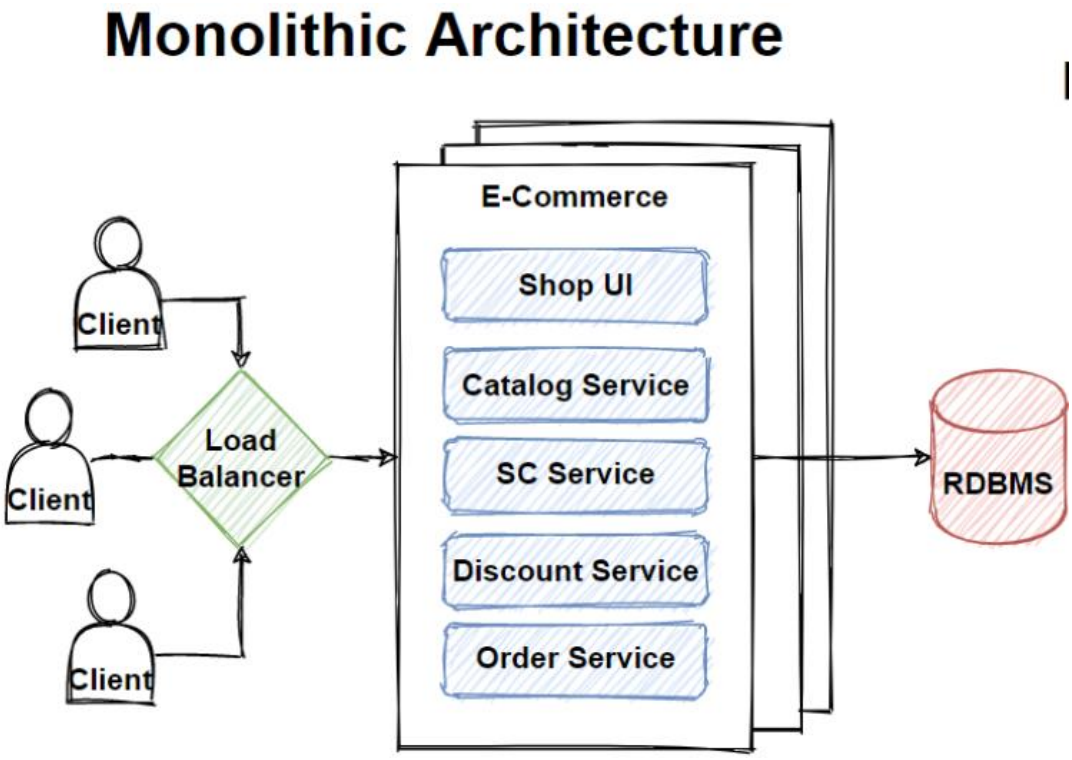


Рис. 2.1 – Схема монолітної архітектури

Монолітна архітектура є одним з найбільш традиційних і поширених підходів до створення програмного забезпечення. У цій архітектурі вся система розробляється як єдине, нероздільне ціле, де всі компоненти і функції інтегровані в одну велику програму або виконуваний файл. Це означає, що всі модулі і компоненти програми, такі як користувацький інтерфейс, логіка бізнесу і доступ до даних, об'єднані в єдину кодову базу.

Переваги монолітної архітектури

- Простота розробки: оскільки всі компоненти знаходяться в одній кодовій базі, розробникам легше зрозуміти і навігаційно переходити між різними частинами системи, простий процес налаштування і

розгортання, оскільки все програмне забезпечення розгортається як одне ціле.

- Цілісність коду: немає необхідності в управлінні розподіленими системами або сервісами, що зменшує складність та витрати на інтеграцію, легше тестувати, оскільки можна використовувати єдине тестове середовище для всього додатку.
- Висока продуктивність: відсутність накладних витрат на міжпроцесну комунікацію, що може бути притаманне розподіленим системам.

Недоліки монолітної архітектури

- Складність масштабування: важко масштабувати окремі компоненти системи, оскільки масштабувати доводиться всю систему як одне ціле, що може бути неефективно, великі монолітні додатки можуть страждати від проблем продуктивності через те, що весь код працює в одному процесі.
- Ускладненість розробки і підтримки: зі зростанням системи, кодова база стає важчою для управління, що ускладнює внесення змін і додавання нових функцій, будь-яка зміна в коді може вплинути на весь додаток, що збільшує ризик появи помилок і складність їх виявлення.
- Тривалі цикли розгортання: будь-яка зміна в коді вимагає повторного розгортання всього додатку, що може бути повільним і ризикованим процесом; у разі збоїв у розгортанні, вся система може бути недоступною.
- Низька гнучкість: важко впроваджувати нові технології або оновлювати існуючі компоненти без значних змін в усьому додатку, мало можливостей для паралельної роботи різних команд над різними частинами системи.

Монолітна архітектура підходить для невеликих додатків або стартапів, які тільки починають свій шлях і потребують швидкого виходу на ринок. Проте, зі зростанням системи і вимог до неї, багато організацій схиляються до переходу на

більш модульні та гнучкі підходи, такі як мікросервіси, які дозволяють краще управляти складністю та масштабованістю великих додатків.

2.2 UML - діаграми

Щоб зрозуміти які дії можуть виконувати користувачі в системі і як ці дії пов'язані між собою, потрібно створити діаграму варіантів використання.

Діаграми варіантів використання є одним із основних інструментів для моделювання систем. Вони використовуються для представлення функціональних вимог до системи з точки зору користувачів і їх взаємодії із системою.

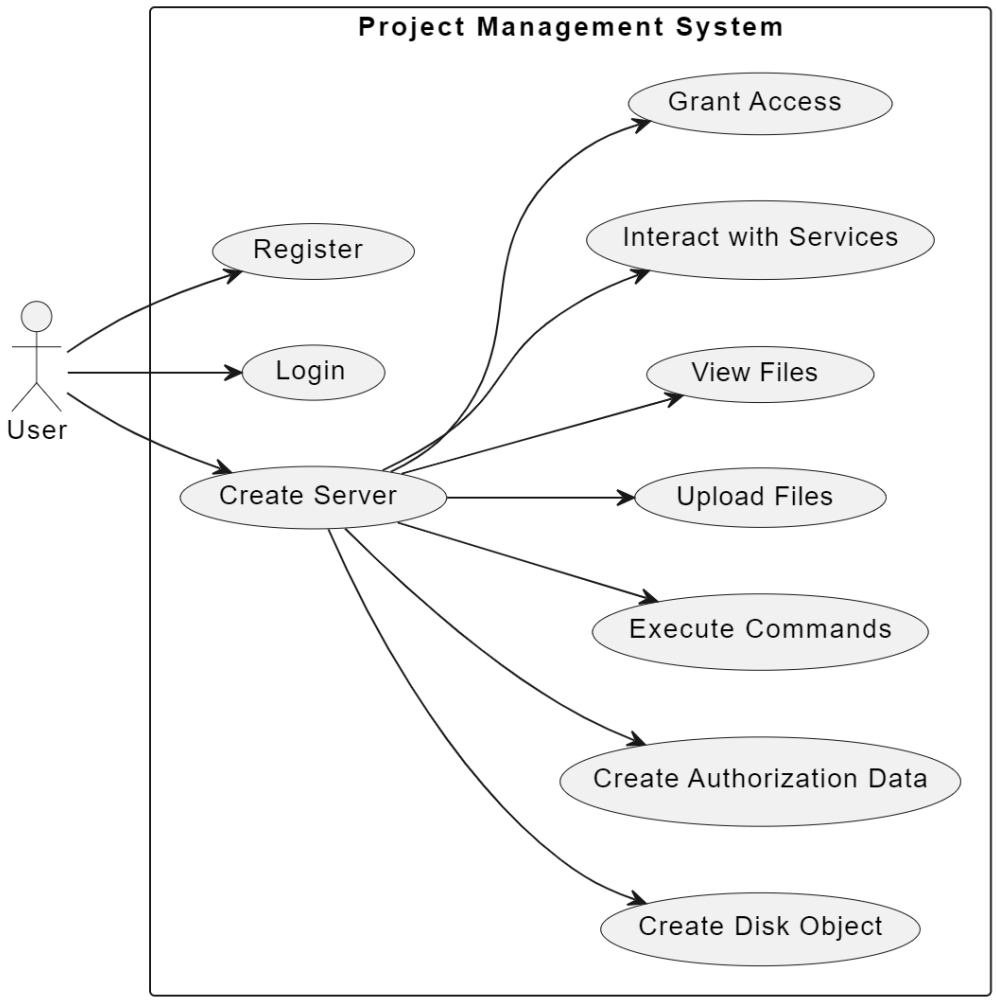


Рис. 2.2 – Діаграма варіантів використання

На діаграмі представлений один актор - Користувач. Йому будуть доступні такі дії як реєстрація в системі і подальший вхід в систему. При реєстрації користувач вказує ім'я, електронну адресу і пароль, які пізніше буде вводити при вході (окрім електронної адреси). Також користувач зможе створювати об’єкт сервера. На сервері він зможе створювати об'єкти «диск», об'єкти даних авторизації, може створити об'єкт «команда» яку потім зможе виконати на цьому сервері. Крім цього користувачу надається можливість завантажувати та переглядати файли, взаємодіяти із сервісами на цьому сервері та надавати доступ іншим користувачам до сервера.

Іншим типом діаграм є діаграми послідовності. Діаграми послідовності є одним з видів діаграм в UML, які використовуються для моделювання взаємодії між об'єктами в системі. Вони показують, як об'єкти обмінюються повідомленнями в певному порядку для виконання певного функціоналу або бізнес-процесу. Ці діаграми дозволяють візуалізувати динамічну поведінку системи, що є важливим для розуміння її роботи на різних етапах розробки.

Діаграми послідовності допомагають моделювати послідовність кроків або дій, які виконуються для досягнення певного результату або завершення бізнес-процесу.

Ці діаграми дозволяють деталізувати вимоги до системи та розробити її структуру, що допомагає визначити, як компоненти повинні взаємодіяти. Вони допомагають розробникам зрозуміти, як об'єкти повинні працювати разом для виконання функцій. Діаграми послідовності забезпечують наочну документацію, яка може бути корисною для розробників, тестувальників та інших учасників проекту. Вони допомагають зрозуміти складні взаємодії в системі, що полегшує підтримку і подальший розвиток системи. Також ці діаграми можуть бути використані для створення тестових сценаріїв, що відображають реальні сценарії використання системи. Це допомагає забезпечити, що всі важливі взаємодії та процеси були враховані і протестовані.

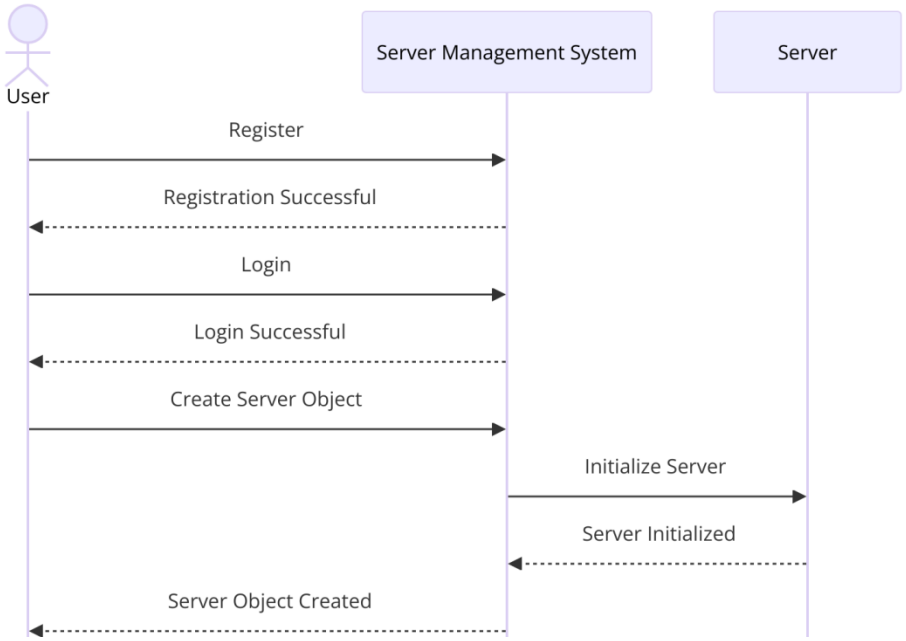


Рис. 2.3 – Фрагмент діаграми послідовностей

Повна діаграма послідовностей знаходиться у Додатку А

На діаграмі послідовності зображена частина функціоналу, а саме те, що Користувач може зареєструватися на сайті після чого йому прийде повідомлення про успішну реєстрацію. Після реєстрації користувач може здійснити вхід у систему і йому також прийде сповіщення, але вже про успішний вхід.

Коли зареєстрований користувач увійшов в систему, він може створити сервер. Якщо сервер був успішно створений йому прийде повідомлення про це. Так само створюються об'єкти дисків та даних авторизації. Також після того як було створено дані авторизації на сервері користувач може виконати команди і завантажити файл на сервер.

Окрім цього користувач може відкрити і переглянути файл який є на сервері, а також, якщо користувач додав в друзі іншого користувача, він може надати йому дозвіл на перегляд або на редагування свого сервера.

Для того щоб структурувати саму систему, тобто код, необхідно побудувати одну із найважливіших діаграм – діаграму залежностей. Діаграми залежностей є важливим типом діаграм у UML, які використовуються для відображення залежностей між різними компонентами системи. Вони схожі до діаграм класів, але крім самих класів показують, як різні частини системи взаємодіють і залежать одна від одної. Ці діаграми допомагають розробникам зрозуміти і керувати

складністю системи, а також забезпечити її модульність і повторне використання компонентів.

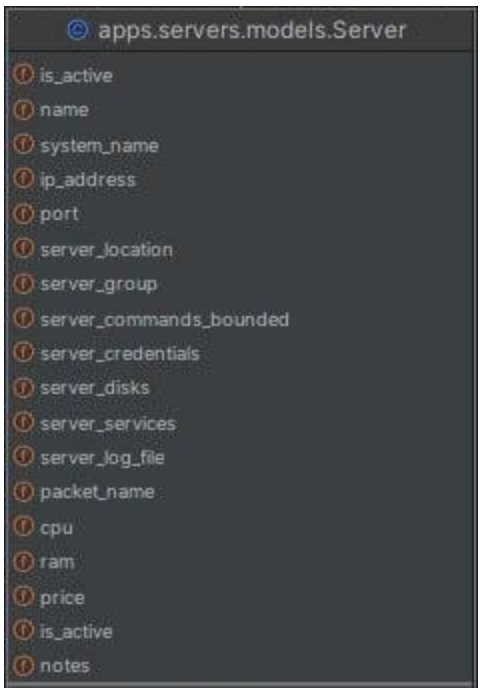


Рис. 2.4 – Фрагмент класу Server

Повна діаграма залежностей знаходиться у Додатку Б, клас Server винесено у додаток В

В моделі сервера є такі поля: ім'я, назва системи, ip-адреса, порт, кількість оперативної пам'яті, ціна сервера. Це поля, що відповідають за опис самого сервера. Також присутні поля які є посиланнями на інакші моделі такі як локація сервера, група сервера, серверні команди, дані входу, диски, сервіси, log файли і т. д.

Взаємодію даної системи з серверами користувача було здійснено за допомогою бібліотеки paramiko. Вона дозволяє посилати сигнали напряму на сервер здійснювати вхід в нього та ще деякі інші функції.

Стосовно методів об’єкта Сервер, то в нього є основні головні методи такі як авторизація та виконання команди. Всі наступні команди і методи використовують попередні основні команди для того щоб виконувати більш специфічний функціонал такий як перевірка наявності файлу на сервері, отримання інформації про диски, увімкнення або вимкнення сервісів.

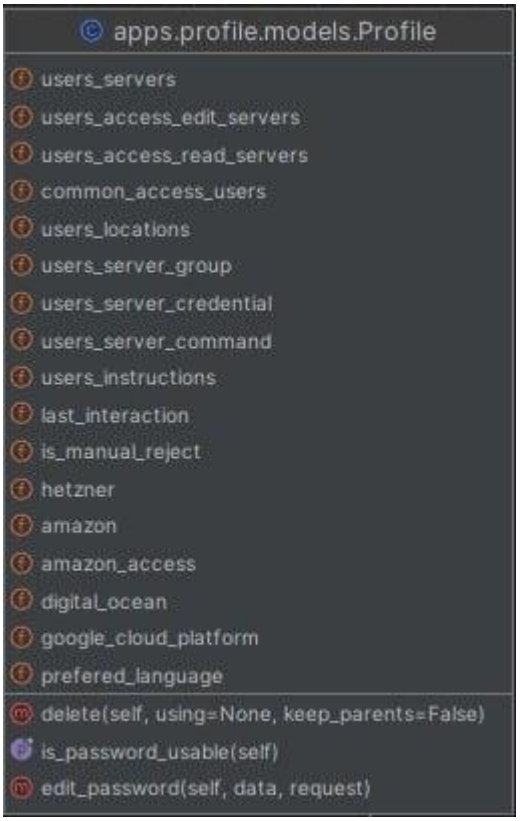


Рис. 2.5 – Модель Профіль

Модель профіля відповідає здебільшого за спільний доступ до серверів. Основна функція в профілі створює користувача та зберігає інформацію про нього. Також вона відповідає за спільний доступ до серверів, тобто надає права користувачам на доступ для редагування попередньої версії сервера, та прив'язку об'єктів до користувача для його безпеки – щоб інший користувач не зміг змінити видалити чи переглянути чужий файл.

В майбутньому також передбачена інтеграція з сторонніми продуктами тому було зроблені поля для API ключів таких як Amazon digital ocean.

2.3 Система контролю версій

Для контролю версій проекту було обрано GitHub, оскільки він є однією з найбільших і найпопулярніших систем для контролю версій. Більшість відкритих проєктів розміщені саме на GitHub, що робить його стандартом у галузі. Крім

того, GitHub пропонує потужні інструменти для спільної роботи, управління кодом і інтеграції з іншими сервісами. Мій особистий великий досвід користування GitHub також став значним плюсом при виборі цієї системи, оскільки це дозволяє ефективно використовувати всі її можливості та забезпечує зручність у роботі з проектом.

Переваги використання GitHub:

- GitHub надає інструменти для командної роботи, такі як pull requests, code reviews, і issue tracking, що спрощує процес спільного розроблення і підтримки коду.
- GitHub забезпечує високий рівень безпеки, включаючи підтримку двофакторної аутентифікації та інструменти для управління доступом до репозиторіїв.
- GitHub інтегрується з багатьма іншими сервісами та інструментами для розробки, такими як CI/CD (Continuous Integration/Continuous Deployment) платформи, системи управління проектами, та інші DevOps інструменти.
- GitHub дозволяє легко створювати і підтримувати документацію для проекту за допомогою Markdown файлів, які можуть бути зберігані в репозиторіях.
- Велика спільнота розробників на GitHub дозволяє легко знайти та використовувати сторонні бібліотеки, отримувати допомогу та ділитися своїм кодом.

GitHub став важливою частиною інструментарію розробників завдяки своїй зручності, надійності та багатству функцій, що робить його ідеальним вибором для контролю версій у цьому проекті.

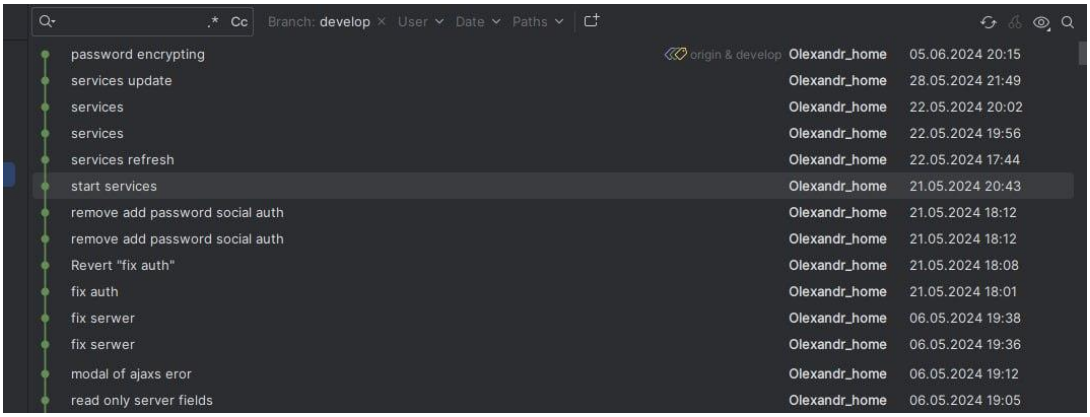


Рис. 2.6 – Фрагмент дерева комітів

2.4 Реалізація проєкту

```
<div class="modal_overlay modal_cancel">
    <div class="my_modal border border-1 rounded-3 border-primary">
        <div class="border border-1 border-primary rounded-circle gap-1 d-flex justify-content-center"
            <img src="/media/icons/green_add_icon.svg" alt="{% trans "Додати" %}">
        </div>
        <div class="fields">
            <form id="add_form" {% if model_name == "Server" %} class="d-grid " {% endif %}>
                {% for form_field in form %}
                    <div class="field-div ">
                        {% if form_field.label == 'Price' ... %}
                        {{ form_field }}
                    </div>
                {% endfor %}
            </form>
        </div>
        <div class="buttons w-100">
            <div type="button" class="modal_submit_add bg-primary text-white text-center rounded-3 p-2
                {% trans "Додати" %}
            </div>
            <div type="button" class="modal_cancel border-1 border border-primary text-center text-prim
                {% trans "Скасувати додавання" %}
            </div>
        </div>
    </div>
</div>
```

Рис. 2.7 - HTML шаблон модального вікна

Зображення показує фрагмент HTML-коду з використанням шаблону Django, який створює модальне вікно для додавання нових елементів. Модальне вікно огорнуте в контейнер з класами modal_overlay та modal_cancel, що забезпечують фон та закриття модального вікна. Внутрішній контейнер з класами my_modal border border-1 rounded-3 border-primary визначає стилізацію

модального вікна. Заголовок модального вікна містить іконку з додатковим текстом "Додати", використовуючи тег <img> для іконки та {% trans "Додати" %} для перекладу.

Форма розміщена в контейнері з класом fields та починається з тега <form id="add_form">, який перевіряє, чи model_name дорівнює "Server" для додавання класу d-grid. Усередині форми використовується цикл {% for form_field in form %}, який ітерує через всі поля форми. Кожне поле форми огорнуте в контейнер з класом field-div, і є умова {% if form_field.label == 'Price' %}, що дозволяє перевіряти мітку поля та відповідно до цього виводити поле форми.

Кнопки для підтвердження та скасування додавання розміщені в контейнері з класом buttons w-100. Перша кнопка має клас modal_submit_add і текст "Додати", а друга кнопка має клас modal_cancel і текст "Скасувати додавання". Код використовує Bootstrap для стилізації та Django шаблони для генерації динамічного контенту та перекладу тексту.

```
// Обробник для кнопки підтвердити додавання
$(document).on('click', '.modal_submit_add', function () {
    const model_name = $(this).data('model_name');
    const url = `/modal_add_object/${model_name}/`;
    const form = $('#add_form');
    const Data = new FormData();
    let formData = form.serialize();
    const file = form.find('#id_ssh_key')[0]
    if (file && file.files) {
        Data.append( name: "form_data",  value: `${formData}&ssh_key=''`);
        const file_obj = file.files[0];
        Data.append( name: "file", file_obj);
    } else {
        Data.append( name: "form_data", formData);
    }
    if (model_name === "ServerCommandBounded" || model_name === "ServerDisk"
        const server_id = $('#server-id').data('server_id');
        Data.append( name: "server_id", server_id);
    }
    let form_id = $(this).data('form_id');

    sendData(url, "POST", Data, form_id);
});
```

Рис. 2.8 – Обробник подій

Зображення показує фрагмент JavaScript-коду з використанням бібліотеки jQuery для обробки події кліку на кнопку підтвердження додавання, яка має клас .modal_submit_add. Коли користувач натискає на цю кнопку, викликається функція, яка починає з отримання назви моделі за допомогою $(this).data('model_name'), потім визначає URL для надсилання даних і отримує форму з ID add_form.

Код створює об'єкт FormData для збору даних і серіалізує дані форми в строку. Далі отримується елемент файлу з форми. Якщо у формі є файл, він додається до FormData. Якщо файл відсутній, додаються серіалізовані дані форми. У випадку, якщо назва моделі відповідає значенням ServerCommandBounded або ServerDisk, до FormData додається ID сервера.

Після цього визначається ID форми, і викликається функція sendData для надсилання даних на сервер з використанням методу "POST". Цей скрипт забезпечує динамічне збирання та надсилання даних форми, включаючи файли, на сервер для обробки.

```python
def post(self, request, model_name):
    form = None
    if model_name in globals():
        form_class = globals()[f"{model_name}Form"]
        form_data = parse_qs(request.POST.get('form_data'))
        form_data = {key: values[0] for key, values in form_data.items()}
        form = form_class(form_data)
    if form.is_valid():
        modal_object = form.save()
        if model_name == "Location":
            request.user.users_locations.add(modal_object)
        elif model_name == "ServerGroup":
            request.user.users_server_group.add(modal_object)
        elif model_name == "Server":
            request.user.users_servers.add(modal_object)
        elif model_name == "ServerCredential":
            file = request.FILES.get('file', None)
            if file:
                now = datetime.now()
                dt_string = now.strftime("%Y_%m_%d_%H_%M_%S")
                modal_object.ssh_key.save(
                    f'admin/ssh_keys/{request.user.id}/{dt_string}_-_{file.name.replace(" ", "_")}', file)
            request.user.users_server_credential.add(modal_object)
        elif model_name == "ServerCommand":
            request.user.users_server_command.add(modal_object)
        elif model_name == "ServerCommandBounded":...
        elif model_name == "ServerDisk":...
        elif model_name == "ServerServices":...
        elif model_name == "LogFile":
            server_id = request.POST.get('server_id')
            server = Server.objects.get(id=server_id)
            ...
            server.server_log_file.add(modal_object)
            server.save()
        request.user.save()
        return JsonResponse({'success': True})
    else:
        ...
        return JsonResponse({'success': False, 'errors': form.errors})
```

Рис. 2.9 – Фрагмент Python коду

Зображення показує фрагмент коду на Python, написаного у стилі Django. Це метод post у вигляді класу, який обробляє HTTP POST запити для збереження даних залежно від моделі.

Метод починається з перевірки, чи є ім'я моделі у глобальному просторі імен. Якщо є, він витягує відповідний клас форми і створює екземпляр цієї форми з даними, отриманими з POST запиту.

Далі метод перевіряє, чи форма є дійсною. Якщо так, збережений об'єкт додається до відповідного користувацького списку залежно від моделі: наприклад, Location, ServerGroup, Server, ServerCredential, ServerCommand, ServerCommandBounded, ServerDisk, ServerServices, або LogFile.

Для моделі ServerCredential додатково обробляються файли: зберігається файл SSH-ключа з назвою, яка включає мітку часу і ID користувача. Для моделі LogFile зберігається ID сервера, до якого прив'язується лог-файл.

Якщо ж форма недійсна, метод повертає JSON-відповідь з успіхом False та помилками форми. В іншому випадку, якщо все пройшло успішно, повертається JSON-відповідь з успіхом True.

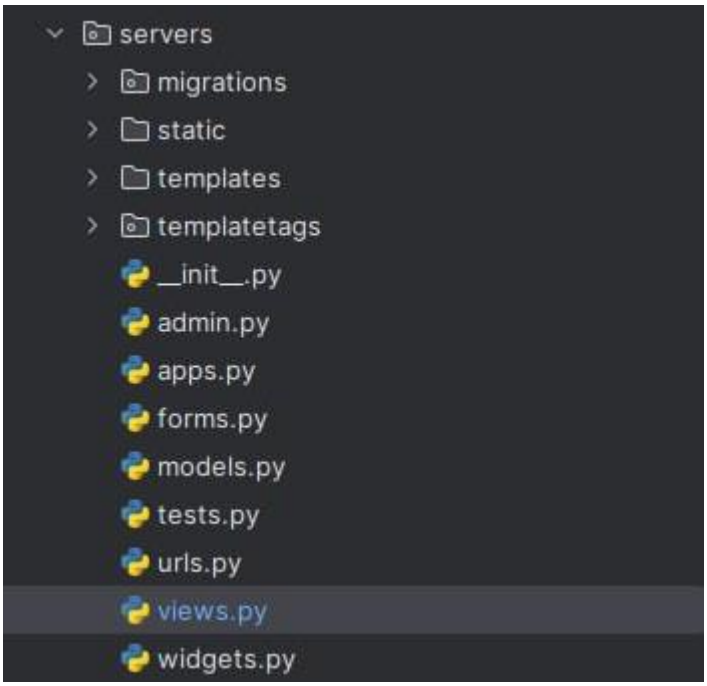


Рис. 2.10 – Фрагмент структури

На цьому скріні містяться структури всіх файлів Django. Вони складаються з таких файлів як urls.py – це файл в якому знаходяться посилання на сторінки або запити, views.py: у ньому знаходяться обробники цих посилань, models.py призначений для опису класів (моделей) на основі яких будуть створені таблиці бази даних, а також методи для них і admin.py: в ньому знаходиться опис моделей для їх відображення в адміністративній панелі проекту.

Окрему увагу можна приділити елементу Термінал, який пов'язує термінал сервера з нашим проектом з моїм проектом. Він був реалізований за допомогою бібліотеки xtrm, а також веб-сокетів.

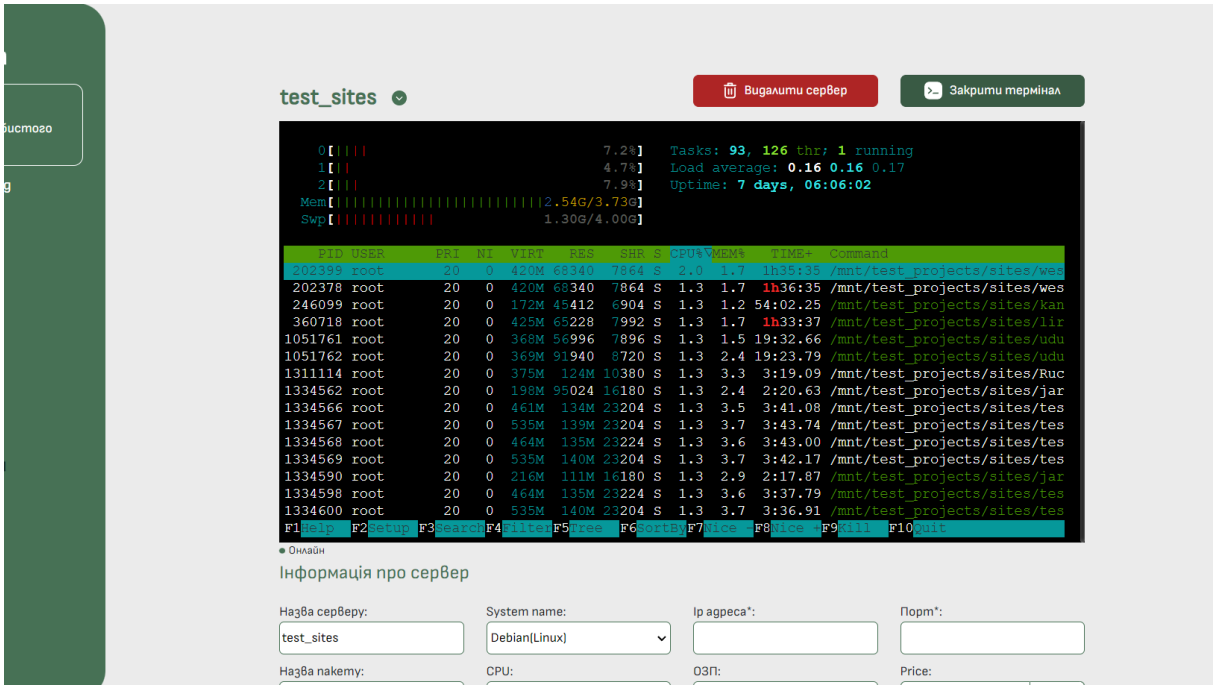


Рис. 2.11 – Вигляд терміналу

xterm.js - це бібліотека JavaScript, яка дозволяє створювати емулятори терміналу у веб-браузерах. Вона підтримує основні функції терміналу, такі як обробка введення та виведення тексту, керування курсором та використання кольорів. xterm.js добре інтегрується з іншими веб-технологіями, що дозволяє розробникам легко додавати термінальні функції до своїх веб-додатків. Вона працює в більшості сучасних браузерів і має підтримку плагінів для розширення своїх можливостей.

Веб-сокети - це протокол комунікації, який забезпечує постійне повнодуплексне з'єднання між веб-браузером і сервером. Це означає, що дані можуть передаватися в обидва боки одночасно, що зменшує затримку в порівнянні з традиційними HTTP-запитами. Веб-сокети ефективні для додатків, які потребують швидкої і постійної передачі даних, таких як чати, онлайн-ігри та інші інтерактивні сервіси.

Поєднання xterm.js та веб-сокетів дозволяє створювати потужні інтерфейси командного рядка у веб-додатках. xterm.js відповідає за відображення та взаємодію з терміналом у браузері, а веб-сокети забезпечують швидку і ефективну передачу даних між браузером та сервером. Це дозволяє реалізувати інтерактивні термінальні сесії, де користувачі можуть вводити команди і отримувати результати в режимі реального часу, працюючи безпосередньо з веб-інтерфейсу.

Завдяки xterm.js користувачі можуть мати повноцінний термінал у браузері, що є особливо корисним для адміністрування серверів або віддаленої роботи з командним рядком. Веб-сокети, у свою чергу, забезпечують безперервний і швидкий обмін даними між клієнтом і сервером, що дозволяє оперативно виконувати команди і отримувати результати. Це робить їх ідеальним вибором для створення інтерактивних і реальних часу веб-додатків.

```
devops-regulus +2 *
$('body').on('click', '#open_terminal:not(.active)', function () : void  {
    $(this).addClass('active');

    1+ usages  psina home
    function term_write(text) : void  {
        if (term) {
            term.write(text);
            if (!term.resized) {
                resize_terminal(term);
                term.resized = true;
            }
        }
    }


    let server_id = $('#server-id').data('server_id');

    let url : string  = `ws://${window.location.host}/ws/ssh/?server_id=${server_id}`;

    let socket : WebSocket  = new WebSocket(url);
    window.socket = socket;
    let term : Terminal  = new Terminal();
    term.open(document.getElementById( elementId: 'terminal'));
```

Рис. 2.12 – Скрипт підключення

На рисунку зображено скрипт який встановлює зв'язок з сервером і проектом через клас який знаходиться на бекенді за допомогою веб-сокетів.

```
class WebSSH(WebsocketConsumer):
    message = {'status': 0, 'message': None}

    Gelbot +1 *
    def connect(self):
        self.accept()
        query_string = self.scope.get('query_string')
        query_params = parse_qs(query_string.decode('utf-8'))
        self.ssh = SSH(websocker=self, message=self.message)
        server_id = query_params['server_id'][0]
        from apps.servers.models import Server
        server = Server.objects.get(id=server_id)
        host = server.ip_address
        port = server.port
        user = server.server_credentials.username
        password = server.server_credentials.password
        ssh_connect_dict = {...}
        self.ssh.connect(**ssh_connect_dict)

    devops-regulus *
    def disconnect(self, close_code):
        self.ssh.close()
    psina home +1 *
    def receive(self, text_data=None, bytes_data=None):
        try:
            msg = json.loads(text_data)
        except Exception as e:
            return
        if not isinstance(msg, dict):
            return
        resize = msg.get('resize', None)
        if resize and len(resize) >= 2:
            self.ssh.resize_pty(*resize)
            return
        data = msg.get('data')
        if data and isinstance(data, str):
            self.ssh.shell(data)
```

Рис. 2.11 – Клас для створення зв’язку з сервером

Цей код описує WebSocket-консюмер у Django для реалізації функціональності SSH через вебсокети. Клас WebSSH наслідує від WebsocketConsumer і використовує бібліотеку paramiko для встановлення SSH-з'єднання. Розглянемо кожен метод детально.

Клас WebSSH наслідує від WebsocketConsumer з бібліотеки Django Channels. Змінна класу message використовується для зберігання стану та повідомлень.

Метод connect викликається, коли клієнт підключається до WebSocket. Викликається self.accept(), щоб прийняти підключення. Отримуємо рядок запиту з

scope та перетворюємо його в параметри. Створюється об'єкт SSH з параметрами з'єднання. Отримуємо дані сервера з бази даних за server_id. Викликається метод connect для встановлення SSH-з'єднання з використанням даних сервера.

Метод disconnect викликається, коли клієнт відключається від WebSocket. Він закриває SSH-з'єднання, викликаючи метод close.

Метод receive обробляє повідомлення, які надходять від клієнта, перетворює текстові дані в JSON, перевіряє, чи є отримане повідомлення словником. Якщо повідомлення містить параметри resize, викликається метод resize_pty. Якщо повідомлення містить команду, передає її до SSH-сесії через метод shell. Цей клас забезпечує інтерактивну SSH-сесію через WebSocket, дозволяючи клієнтам виконувати команди на віддалених серверах в режимі реального часу.

2.5 Тестування системи

Тестування програмного забезпечення (ПЗ) є критично важливою частиною процесу розробки, яка спрямована на забезпечення якості, надійності та відповідності системи вимогам. Його основна мета - виявити та виправити помилки до випуску продукту, мінімізуючи ризики для користувачів і бізнесу.

Процес тестування починається з визначення вимог до ПЗ. На основі цих вимог створюються тестові випадки, які перевіряють різні аспекти роботи системи. Цей процес може включати ручне тестування, де інженери вручну перевіряють роботу системи, і автоматизоване тестування, де використовуються спеціальні інструменти для автоматичного виконання тестів.

Тестування поділяється на кілька основних типів. Функціональне тестування перевіряє, чи відповідає ПЗ функціональним вимогам. Це включає перевірку окремих функцій, інтеграцію компонентів та поведінку системи в цілому. Наприклад, у веб-додатку може бути перевірено, чи правильно працює форма реєстрації користувача або чи коректно виконується пошук по сайту.

Функціональне тестування є важливим для забезпечення того, щоб система виконувала свої завдання відповідно до специфікацій.

Таблиця 2.1 – Результати тестування сервісу

Тип тестування	Результати тестування
Функціональне	Функціональне тестування підтвердило, що всі основні функції проекту, такі як створення користувачів та управління базами даних, працюють відповідно до вимог і забезпечують коректну взаємодію з користувачами.
Навантажувальне (Blazemeter)	Навантажувальне тестування показало, що система здатна стабільно обробляти велику кількість одночасних запитів користувачів без зниження продуктивності, забезпечуючи високу швидкість відповіді.
Автоматизоване (Selenium)	Успішне автоматизоване тестування для даного проекту забезпечило швидку перевірку всіх основних функцій після кожної зміни в коді, що дозволило швидко виявляти та виправляти помилки без значних витрат часу.

Навантажувальне тестування оцінює, як система працює під навантаженням. Це включає перевірку швидкості відповіді системи, її стабільності та здатності витримувати великі обсяги даних та високе навантаження. Наприклад, може бути перевірено, як веб-сайт працює при одночасному доступі тисячі користувачів, або як швидко база даних обробляє великі запити. Це особливо важливо для веб-додатків та інших систем, які обробляють велику кількість запитів від користувачів.

Автоматизоване тестування використовує спеціальні інструменти для автоматичного виконання тестів. Це дозволяє значно прискорити процес тестування та підвищити його ефективність. Автоматизовані тести можуть бути запущені багаторазово без участі людини, що зменшує ймовірність помилок та дозволяє швидко перевіряти систему після кожної зміни в коді. Це є особливо

корисним при регресійному тестуванні, коли необхідно перевірити, що нові зміни не призвели до виникнення нових помилок у вже працюючих функціях.

Таким чином, тестування програмного забезпечення є комплексним процесом, який охоплює різні методи і типи перевірок для забезпечення високої якості та надійності продукту. Це важливий етап розробки, який допомагає виявляти та виправляти помилки, забезпечуючи позитивний досвід користувачів та досягнення бізнес-цілей.

Наразі проект запущений у режимі бета-тестування і знаходиться в користуванні 10 користувачів.

Розробка та тестування системи пройшли успішно, що підтвердило її готовність до реальної експлуатації. Використання технологій Python та Django дозволило швидко і ефективно створити функціональний та масштабований продукт. Успішне автоматизоване тестування забезпечило стабільність системи після кожної зміни в коді, дозволяючи швидко виявляти та виправляти помилки. Навантажувальне тестування продемонструвало здатність системи обробляти велику кількість одночасних запитів користувачів без зниження продуктивності, що є критично важливим для її надійності та доступності. Функціональне тестування підтвердило, що всі основні функції проекту працюють відповідно до вимог, забезпечуючи коректну взаємодію з користувачами. Таким чином, реалізація проекту показала, що система відповідає високим стандартам якості, надійності та ефективності, що є запорукою її успішного впровадження та використання.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

3.1 Долікарська допомога при переломах

Долікарська допомога – це сукупність медичних заходів, які надаються при невідкладних станах до прибуття лікаря. Ці заходи можуть бути необхідні у різних ситуаціях, таких як виробничі травми, побутові нещасні випадки, дорожньо-транспортні пригоди, катастрофи та техногенні аварії. Крім того, долікарська допомога потрібна при гострих неврологічних, терапевтичних, хірургічних та термінальних станах. Відсутність своєчасної долікарської допомоги при нещасних випадках або раптових гострих захворюваннях може призвести до тяжких наслідків, включаючи летальні випадки. Своєчасно надана долікарська допомога має велике значення для подальшого лікування постраждалих та хворих. Вона сприяє скороченню термінів їх медичної та трудової реабілітації.

Перелом – це порушення цілісності кісток. Переломи поділяються на травматичні та патологічні, закриті (без пошкодження шкіри) і відкриті (з пошкодженням шкіри в зоні перелому). Відкриті переломи особливо небезпечні через ризик інфікування уламків, що може призвести до остеомієліту. Переломи можуть бути повними та неповними. Неповні переломи характеризуються тріщинами, коли порушується тільки частина кістки. За формою переломи поділяються на поперечні, косі, спіральні, осколочні, від стиснення, компресійні тощо. Можливе зміщення кісткових уламків під кутом, по довжині або бокове зміщення.

Основні симптоми переломів включають різкий біль, порушення функції ураженої ділянки, набряк та крововилив у зоні перелому, вкорочення кінцівки та ненормальну патологічну рухомість кістки. Часто спостерігається нерівність кісток, хрумтіння при натисканні, а при відкритому переломі – виступання уламка кістки.

Заходи долікарської допомоги при переломах включають фіксацію кісток в області перелому, протишокові заходи та транспортування постраждалого до

медичного закладу. Основне завдання – забезпечити нерухомість пошкоджених кісток, суглобів та кінцівок у найбільш зручному для постраждалого стані. Іммобілізація допомагає зменшити біль, що є ключовим засобом запобігання шоку. Найчастіше трапляються переломи кінцівок. Правильна фіксація пошкоджених кінцівок запобігає зміщенню уламків, зменшує пошкодження судин, нервів, м'язів та шкіри гострими краями уражених кісток [7].

Для фіксації кінцівок використовують транспортні шини з підручного твердого матеріалу. Кінцівки біля рани або перелому обробляють йодом або антисептиком, а при відкритому переломі накладають асептичну пов'язку. Під час надання допомоги не слід намагатися визначити наявність перелому шляхом мацання місця ушкодження або примушування постраждалого рухати, піднімати або згинати кінцівку. Такі дії можуть різко підсилити біль, спричинити зміщення та додаткове ушкодження м'яких тканин.

Для забезпечення нерухомості зламаної кінцівки застосовують спеціальні дротяні або фанерні (дерев'яні) шини. Шина повинна бути накладена так, щоб надійно іммобілізувати два сусідні з місцем ушкодження суглоби (вище і нижче), а при переломах плеча або стегна – три суглоби. Шину накладають поверх одягу або кладуть під неї щось м'яке, як-от вату, шарф або рушник. Накладену шину необхідно прикріпити до кінцівки бинтом, рушником або ременем. Як шину можна використати дошку, палицю, лижу тощо. Імпровізовану шину прикладають з двох протилежних сторін уздовж ушкодженої кінцівки та обгортають бинтом. Шина повинна бути накладена так, щоб її центр знаходився на рівні перелому, а кінці накладалися на сусідні суглоби по обидві сторони перелому.

Фіксація відкритого перелому потребує дотримання додаткових умов: не можна накладати шину безпосередньо на місце відкритого перелому. Потрібно перебинтувати кінцівку поверх одягу або взуття та підкласти під шину щось м'яке, попередньо зупинивши кровотечу. Під час транспортування шину надійно закріплюють, щоб зафіксувати область перелому. Під шину підкладають вату або тканину та фіксують два суглоби вище і нижче перелому. Правильна фіксація запобігає шоку [2].

3.2 Естетичне оформлення та ергономічне дослідження робочого місця оператора

Проектування та розробка веб-додатків вимагає постійної роботи з ПК. Тому важливо правильно організувати робоче місце оператора ПК, оскільки ергономічно оптимізована робоча зона сприяє збереженню здоров'я та комфорту та підвищує продуктивність праці.

Ергономіка вивчає, розробляє та рекомендує проектування, виготовлення та експлуатацію технічних засобів, що забезпечують людині необхідний комфорт під час праці та підтримують фізичну форму та працездатність

Робочі місця з комп'ютерами повинні відповідати вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце під час виконання робіт стоячи» та ДСанПІН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Основні положення:

- Конструкція робочого місця та взаємне розташування всіх його елементів (сидінь, органів управління, індикаторів тощо) повинні відповідати антропометричним, фізіологічним і психологічним вимогам і виду виконуваної роботи.
- Конструкція робочого місця повинна забезпечувати надійне виконання робочих процесів у зоні руху.
- Висота сидіння і підставки для ніг (якщо робоча поверхня не регулюється по висоті) повинна регулюватися.
- Висота робочої поверхні визначена за номограмою для робітника зростом 1800 мм.
- Оптимальна робоча поза для невеликих працівників досягається збільшенням висоти робочого сидіння та підставки для ніг на величину, що дорівнює різниці між висотою робочої поверхні працівника з ростом 1800 мм та висотою, яка оптимальна для даного працівника.

- Підніжки повинні регулюватися по висоті. Ширина повинна бути не менше 300 мм, а довжина не менше 400 мм. Поверхня підставки повинна бути рифлена [3].

Основні положення ДСанПІН 3.3.2.007-98:

- Площа робочого місця повинна бути не менше 6,0 кв. м, об'єм – не менше 20,0 кв. м.
- Робоча кімната повинна мати природне та штучне освітлення. Природне освітлення повинно забезпечуватися переважно вікнами, що виходять на північ або північний схід, з коефіцієнтом природного освітлення (КПО) не менше 1,5%.
- Внутрішнє оздоблення будівлі використовувати дифузно-відбиваючі матеріали з коефіцієнтом відбиття стель 0,7-0,8 і стін 0,5-0,6.
- Яскравість ламп загального освітлення в поздовжній і поперечній площинах в діапазоні кутів променя від 50 до 90 градусів від вертикалі повинна бути не більше 200 кд/м², а захисний кут лампи - не менше 40°.
- При установці робочого столу відстань між бортами повинна бути 1,2 м, а відстань від спинки столу до ширми іншого столу - 2,5 м.
- Показник засліплення у разі використання джерел загального штучного освітлення у виробничих приміщеннях має не перевищувати 20.
- Необхідно обмежити нерівномірний розподіл яскравості в полі зору людини, яка оперує екраном.
- Співвідношення яскравості робочої поверхні не повинно перевищувати 3: 1, а співвідношення яскравості робочої поверхні та таких поверхонь, як стіни, обладнання тощо, не повинно перевищувати 5: 1 [10].

Важливим фактором для здоров'я і благополуччя співробітників є правильне розташування комп'ютера, монітора, клавіатури і миші. Висота монітора повинна бути такою, щоб верх був на рівні очей або трохи нижче. Відстань між очима та екраном має бути 50-70 см. Клавіатуру слід розташувати так, щоб плечі були розслаблені, а лікті розташовувалися під прямим кутом.

Щоб забезпечити комфорт і продуктивність на робочому місці, слід враховувати такі фактори:

- Природне та штучне світло має бути достатньо яскравим, але не сліпучим.
- Температура і вологість для офісних приміщень - температура 20-24 °C, відносна вологість 40-60%.
- Для комфортної роботи рівень шуму в приміщенні не повинен перевищувати 50 дБ.
- Свіже повітря та належна вентиляція важливі для підтримки працездатності та запобігання втоми.
- Столи та стільці повинні регулюватися по висоті, щоб забезпечити оптимальне робоче положення.

Тому правильне проектування робочого місця оператора ПК відповідно до вимог ергономічних і санітарних норм є важливою умовою збереження здоров'я, підвищення продуктивності праці та профілактики професійних захворювань.

ВИСНОВКИ

В ході виконання дипломної роботи було здійснено повний цикл розробки програмного забезпечення, включаючи аналіз вимог, проектування архітектури, реалізацію та тестування системи. Основним завданням було створення функціональної та ефективної системи, яка відповідала б потребам користувачів та забезпечувала високу якість роботи.

На етапі аналізу вимог були визначені функціональні та нефункціональні вимоги до системи. Функціональні вимоги охоплювали основні функції, які система повинна виконувати, включаючи управління користувачами, базами даних, генерацію звітів та інтеграцію з іншими системами. Нефункціональні вимоги стосувалися продуктивності, безпеки, масштабованості, надійності та зручності використання.

Для реалізації проекту було обрано технології Python та фреймворк Django, що дозволило швидко і ефективно створити функціональний та масштабований продукт. Використання HTML5, JavaScript, jQuery та Bootstrap для фронтенду забезпечило створення зручного та естетичного інтерфейсу користувача. Використання системи контролю версій GitHub дозволило ефективно керувати змінами в коді та співпрацювати з іншими розробниками.

Процес розробки здійснювався відповідно до методології Agile, що забезпечило гнучкість у управлінні проектом, швидке реагування на зміни вимог та залучення користувачів до процесу розробки. Це сприяло підвищенню якості продукту та задоволенню потреб кінцевих користувачів. Регулярні спринти та ітеративний підхід допомогли постійно вдосконалювати продукт та швидко адаптуватися до змін.

Тестування системи включало автоматизоване, навантажувальне та функціональне тестування. Автоматизоване тестування дозволило забезпечити стабільність системи після кожної зміни в коді, швидко виявляючи та виправляючи помилки. Навантажувальне тестування продемонструвало здатність системи обробляти велику кількість одночасних запитів користувачів без

зниження продуктивності, що є критично важливим для забезпечення надійної роботи в умовах високого навантаження. Функціональне тестування підтвердило, що всі основні функції проекту працюють відповідно до вимог, забезпечуючи коректну взаємодію з користувачами та виконання необхідних операцій.

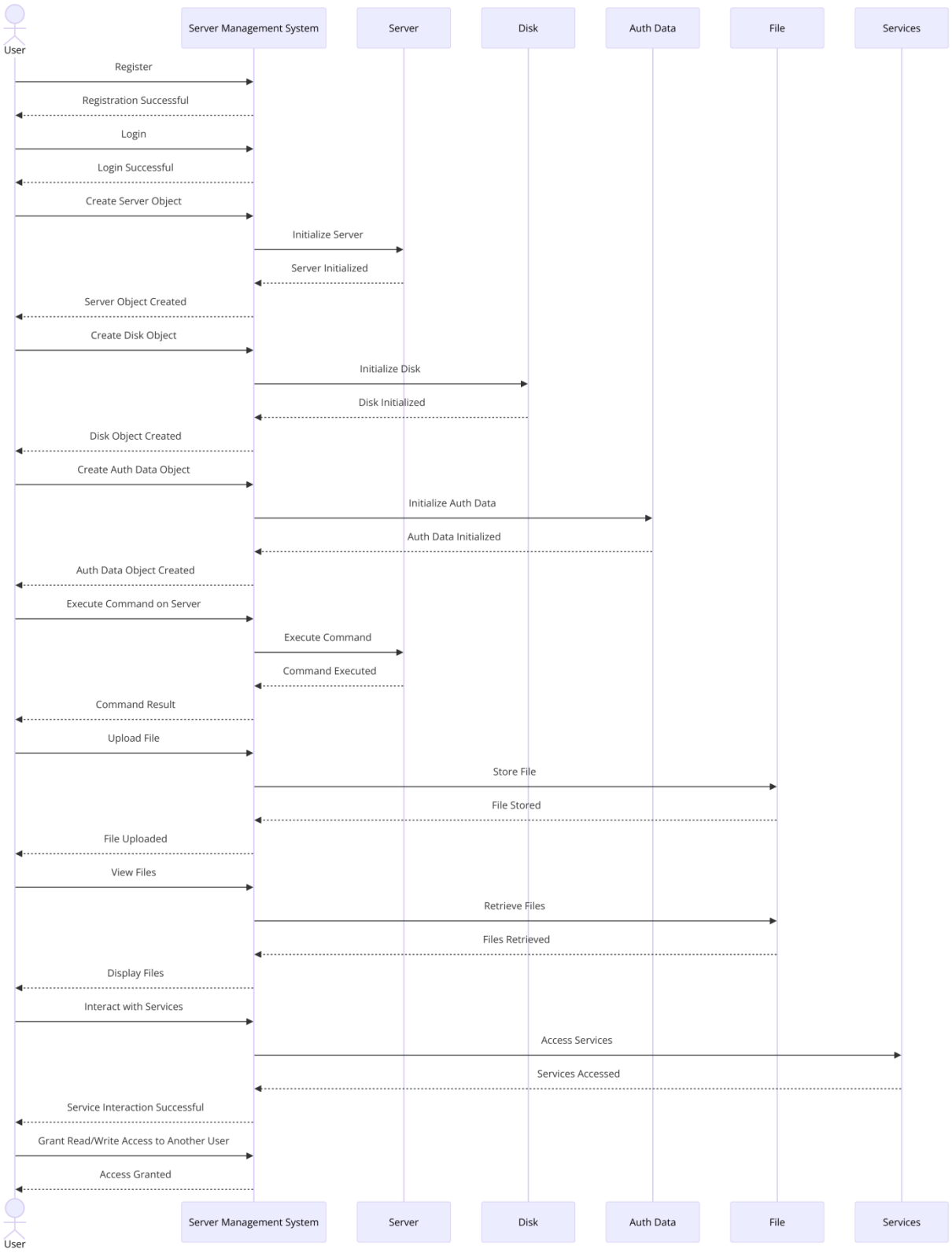
Таким чином, реалізація дипломного проекту показала, що створена система відповідає високим стандартам якості, надійності та ефективності, що є запорукою її успішного впровадження та використання. Виконана робота підтвердила актуальність обраних технологій та методів розробки, а також продемонструвала їх ефективність у вирішенні завдань проекту. Завдяки комплексному підходу до розробки, ретельному тестуванню та впровадженню сучасних технологій вдалося створити продукт, який задовольняє потреби користувачів та відповідає бізнес-цілям.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Маттес Е. Пришвидшений курс Python / Ерік Маттес. – Львів: Видавництво старого лева, 2021. – 600 с.
2. Форсьє Д. Django. Розроблення вебзастосунків на Python / Джефф Форсьє., 2016. – 456 с.
3. Hetzner [Електронний ресурс] – Режим доступу до ресурсу: https://www.hetzner.com/
4. Коул Р. Блискучий Agile / Роб Коул. – Київ: Фабула, 2020. – 192 с.
5. Сесіл Р. Чиста архітектура / Роберт Сесіл. – Київ: Фабула, 2019. – 368 с.
6. Елсон Р. Догоспітальна допомога при травмах / Р. Елсон, Д. Кемпбел. – Київ: Медицина, 2023. – 464 с.
7. Халмурадов Б. Медицина надзвичайних ситуацій / Б. Халмурадов, П. Волянський. – Харків: Центр учбової літератури, 2021. – 208 с.
8. Грибан В. Охорона праці / В. Грибан, О. Негодченко. – Харків: Центр учбової літератури, 2021. – 280 с.
9. Хмельовський В. Охорона праці. Навчальний посібник для студентів ОС «Бакалавр» / В. Хмельовський, М. Мотрич. – Харків: Центр учбової літератури, 2023. – 594 с.
10. Атаманчук П. Безпека життєдіяльності. Навчальний посібник / П. Атаманчук, В. Мендерецький. – Харків: Центр учбової літератури, 2020. – 276 с.
11. Мудрик І. Modeling of adsorption and desorption of hydrocarbons in nanoporous catalytic zeolite media using nonlinear Langmuir isotherm / Іван Мудрик. – Тернопіль, 2019. – 42 с.
12. Мудрик І. High-performance Analyzing Methods for Tremorobjects Abnormal States of Neuro-biosystems with Cognitive Feedbacks / Іван Мудрик. – Тернопіль, 2020. – 268 с.

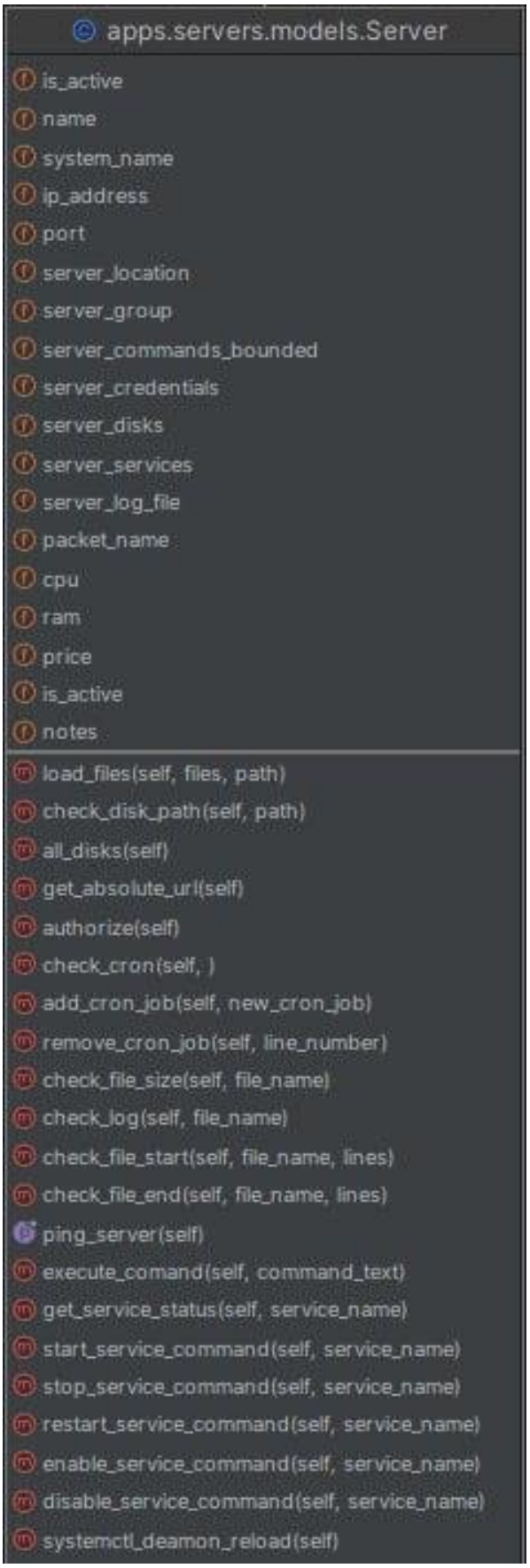
ДОДАТКИ

Додаток А – Діаграма послідовності



Додаток Б - Діаграма залежностей

Додаток В – Клас Server



Додаток Г – Теза для конференції

УДК 004.41
Лань О. – ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА СИСТЕМИ УПРАВЛІННЯ І МОНІТОРИНГУ ПРОЕКТІВ НА СЕРВЕРАХ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ Python, Django

Науковий керівник: д. ф., с. в. Мудрик І. Я.

Marchuk D.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF PROJECT MANAGEMENT AND MONITORING SYSTEM ON SERVERS USING Python, Django

Supervisor: PhD, Senior Lecturer Mudryk I. Y.

Ключові слова: веб-розробка, сервери, ефективність, Python, Django
Keywords: web-development, servers, efficiency, Python, Django

Управління проектами - це складний процес, який потребує ретельного планування, координації та моніторингу. З розвитком технологій з'явилися нові можливості для автоматизації цих процесів, що призвело до зростання популярності систем управління і моніторингу проектами.

Django - це веб-фреймворк Python, який використовується для створення динамічних веб-сайтів та веб-додатків. Django має ряд переваг, які роблять його ідеальною платформою для розробки СУП, зокрема:

Повна стекова природа: Django пропонує комплексне рішення для розробки веб-додатків, включаючи ORM, систему шаблонів, адміністративну панель та багато іншого.

Безпека: Django має вбудовані функції безпеки, які допомагають захистити веб-додаток від кіберзагроз.

Масштабованість: Django може обробляти великі навантаження та підтримувати велику кількість користувачів.

Спільнота: Django має велику та активну спільноту розробників, що означає, що ви можете легко знайти допомогу та підтримку.[1]

Розробка системи управління проєктами на серверах на Django може допомогти командам значно покращити свою роботу. Django - це потужний та гнучкий фреймворк, який може бути використаний для створення системи , яка відповідає потребам будь-якої команди.

Література

Розробка сайтів на Python Django[Електронний ресурс] – Режим доступу до ресурсу: Розробка сайтів та застосунків на Python | Створення сайту на Django.

Додаток Д - Диск