МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
15 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія проєктування системи для підтримки проєктної діяльності з використанням хмарних сервісів»
здобувача групи ІН.м-22 Гайдабруса Олексія Андрійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Олексій ГАЙДАБРУС
(підпис)

Керівник
асистентка кафедри комп'ютерних наук,
к.ф.-м.н. Ольга ШУТИЛЄВА _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН.м-22 Гайдабруса Олексія Андрійовича

1. Тема роботи: «Інформаційна технологія проєктування системи для підтримки проєктної діяльності з використанням хмарних сервісів»
затверджую наказом по СумДУ від «06» грудня 2023 р. №1412-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 15 грудня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для підтримки проєктної діяльності. 3) Розробка інформаційної технології для підтримки проєктної діяльності з використанням хмарних сервісів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» листопада 2023 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для підтримки проєктної діяльності*		
3	*Розробка інформаційної технології для підтримки проєктної діяльності з використанням хмарних сервісів*		
4	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 68 стор., 21 рис., 7 таблиць, 1 додаток, 26 джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню важливої практичної задачі з автоматизації проєктної діяльності.

Об'єкт дослідження – процес підтримки проєктної діяльності.

Мета роботи – розробка інформаційної технології проєктування системи для підтримки проєктної діяльності з використанням хмарних сервісів.

Результати – розроблено інформаційну технологію яка підтримує процес проєктної діяльності з використанням Kanban методології та надає функціонал сповіщень користувачеві системи. Проведено тестування функціональних можливостей розробленого веб-додатку.

ВЕБ-ДОДАТОК, ІНФОРМАЦІЙНА СИСТЕМА, ПІДТРИМКА ПРОЄКТНОЇ ДІЯЛЬНОСТІ, JAVA, KANBAN, SPRING

ЗМІСТ

ВСТУП

Актуальність. Упродовж тривалого часу людству не вдавалося підвищити продуктивність проектної діяльності, а винайдення бумаги було найбільшим стрибком у розвитку цієї сфери. З появою комп'ютерів та інформаційних систем ефективність та швидкість проектної діяльності зросла в рази. Незважаючи на широкий асортимент розроблених рішень, попит на інформаційні системи підтримки проектної діяльності залишається високим. Головною причиною цього є стрімкий розвиток світової економіки та поява нових типів проектів пов'язаних з розробкою програмного забезпечення. Робота в проєкті зі створення інформаційних систем потребує чіткої кооперації розробників між собою. Тому всі компанії, які пов'язані в тій чи іншій мірі з проєктною діяльністю зацікавлені у вирішенні цих задач.

Об'єктом дослідження є проєктна діяльність у компаніях у сфері розробки програмного забезпечення та інших сферах потребуючих автоматизацію.

Предметом дослідження виступають технології автоматизації ведення проєктної діяльності.

Гіпотеза. Враховуючи попит на системи з підтримки проєктної діяльності, доцільною буде розробка інформаційної технології проєктування системи для підтримки проєктної діяльності з використанням хмарних сервісів. Дана інформаційна система зможе вирішити проблеми синхронізації проектних робіт за рахунок використання сучасних підходів в управлінні проектами та підтримки хмарних сервісів для автоматизації проєктної діяльності.

Наукова новизна. Дослідження вносить вклад у розвиток систем автоматизації проектної діяльності та надає можливі варіанти вирішення актуальних проблем.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Дослідження предметної області

Існує багато підходів та методів оптимізації проєктної діяльності. У сфері розробки програмного забезпечення досить популярними є методики Scrum [1, 2] та Kanban [3, 4]. Вони використовуються для побудови шляху роботи над проєктом та підтримки учасників в процесі. Кожен підхід має свої переваги та особливості, тому необхідно визначити який саме підхід треба обрати для розробки інформаційної системи з підтримки проєктної діяльності.

Методологія роботи над проєктом Scrum є чітко врегульованою, робота над проєктом має визначену дату початку та закінчення. Процес роботи над проєктом ділять на спринти, для яких конкретно визначені цілі та задачі. Серед переваг у роботі зі спринтами є їх прогнозованість, адже провести точне планування робіт опираючись на дати дуже легко, та зручні в аналізі, ефективність спринтів зручно порівнювати між собою.

Частковою протилежністю Scrum підходу є методологія Kanban, яка базується на ідеях балансу. Вона поєднує команду розробників в одному неперервному процесі роботи над проєктом, тому інтеграцію цієї технології можна провести на будь-якому етапі роботи. Вся робота поділяється на задачі які можуть перебувати в різних стадіях, наприклад: планується, розробляється тестується чи завершено, а самі задачі розташовуються на Kanban дошці по якій вони рухаються в процесі свого життєвого циклу, на рисунку 1.1 можемо побачити приклад Kanban дошки.

Головним критерієм ефективності в Kanban є середня швидкість проходження задачі по Kanban дошці, яка є прямо пропорційною злагодженості роботи в команді. Ураховуючи велику різноманітність проєктних діяльностей, а не лише розробка програмних продуктів, доцільним буде використати більш гнучку Kanban методику при розробці інформаційної технології проєктування системи для підтримки проєктної діяльності.

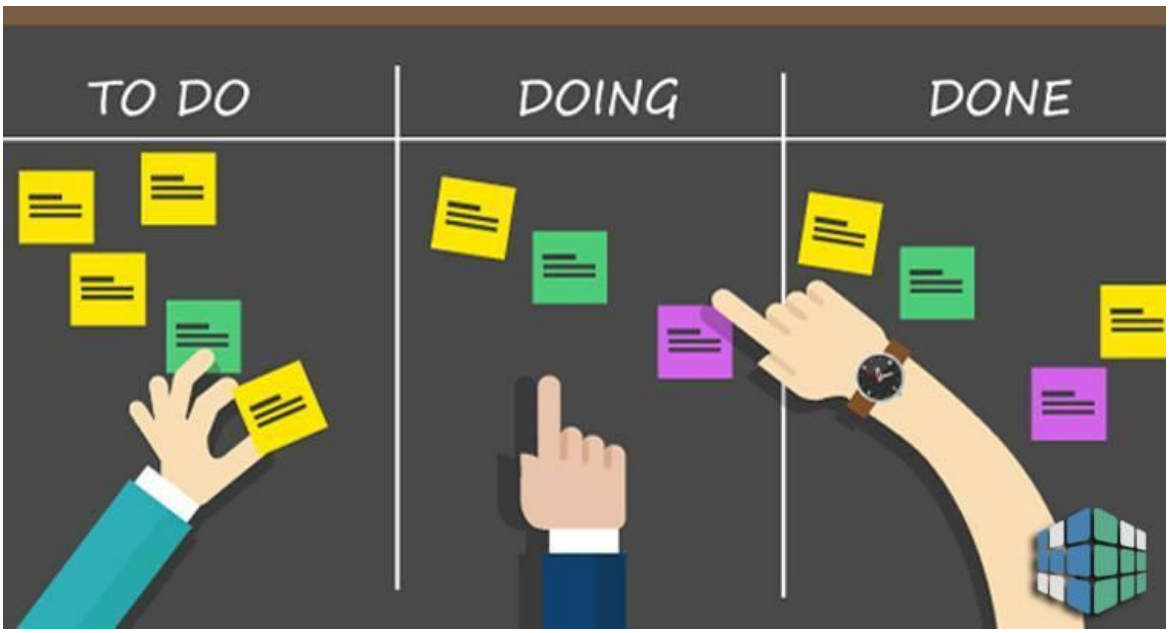

Рисунок 1.1 – Kanban дошка

Високого рівня ефективності можна добитись за рахунок поєднання перевіреної часом технології ведення проєктних робіт та хмарних сервісів. Під формулюванням хмарні сервіси мається на увазі технології інформування виконавців проєктних робіт у реальному часі з використанням мережі інтернет. Системою для інформування користувача було обрано соціальну мережу якою користується близько 50% українців – Telegram [5]. Враховуючи широку популярність доцільно буде розробити систему сповіщення саме в цьому додатку, адже вона буде зрозуміла широкому колу користувачів не залежно від їх рівня знань в IT галузі.

1.2 Постановка задачі

Метою роботи є автоматизація процесу проєктної діяльності за рахунок розробки інформаційної системи підтримки проєктної діяльності з використанням хмарних сервісів.

Для досягнення поставленої мети треба вирішити наступні задачі:

− проаналізувати існуючі технології розробки та обрати оптимальну для реалізації проєкту;

− виконати аналіз аналогів;

− спроєктувати архітектуру інформаційної системи;

− розробити систему сповіщень до інформаційної системи;

− налаштувати хмарне середовище;

− протестувати роботу розробленого web-додатку.

Цільовою аудиторією використання розробленої інформаційної системи є підприємства зацікавлені в підвищені ефективності та швидкості проєктних робіт.

1.3 Огляд аналогів

На ринку представлено велику кількість додатків для підтримки роботи над проєктом, серед яких можна відзначити два популярних сервіси Trello та Notion. Незважаючи на схожість, ці системи мають низку відмінностей які необхідно відзначити.

Система підтримки проєктної діяльності Trello базується на методології дошки Kanban [6]. Додаток Trello підтримує широкий функціонал зі створення та налаштування дошок проєктів, що надає можливості розділяти їх для різних користувачів та делегувати привілеї у створенні нових задач. Також вагомою перевагою є можливість інтеграції з різними сервісами такими як: Evernote, Github, Google Drive, Dropbox та багато інших. Приклад інтерфейсу Trello зображено на рисунку 1.2.

Серед недоліків додатку Trello можна відзначити відсутність підтримки роботи з великими файлами та обмеженість функціоналу без проведення додаткових інтеграцій з іншими сервісами.

У свою чергу Notion є багатофункціональним сервісом, який вирішує великий перелік задач [7]. За допомогою нього можна вести власті плани та нотатки, контролювати та керувати проєктним процесом та навіть налагодити дистанційне навчання. Основною функціональною відмінністю від Trello є система блокноту зі сторінками. На ці сторінки можна додавати будь-який мультимедійний контент для організації планів. За рахунок підтримки

широкого кола можливостей, сервіс Notion здатен замінити більшість сучасних інструментів наприклад: Evernote, Trello чи Google Docs. Інтерфейс сервісу Notion зображено на рисунку 1.3.

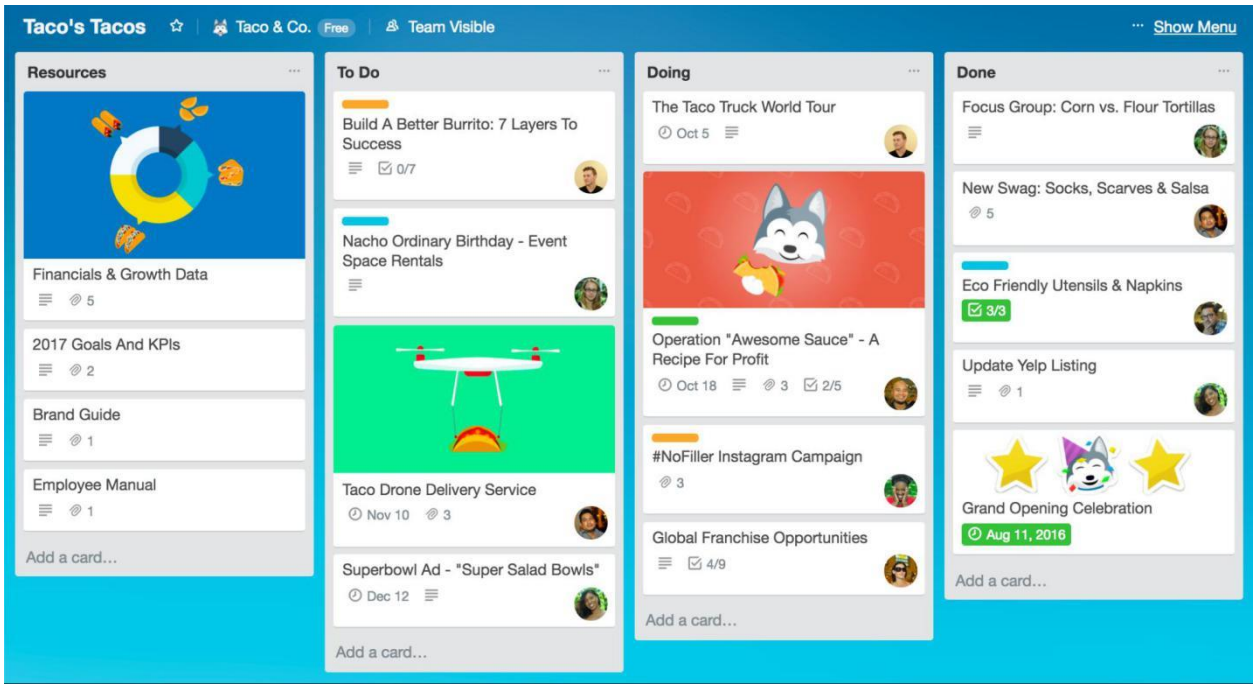

Рисунок 1.2 – Інтерфейс Trello

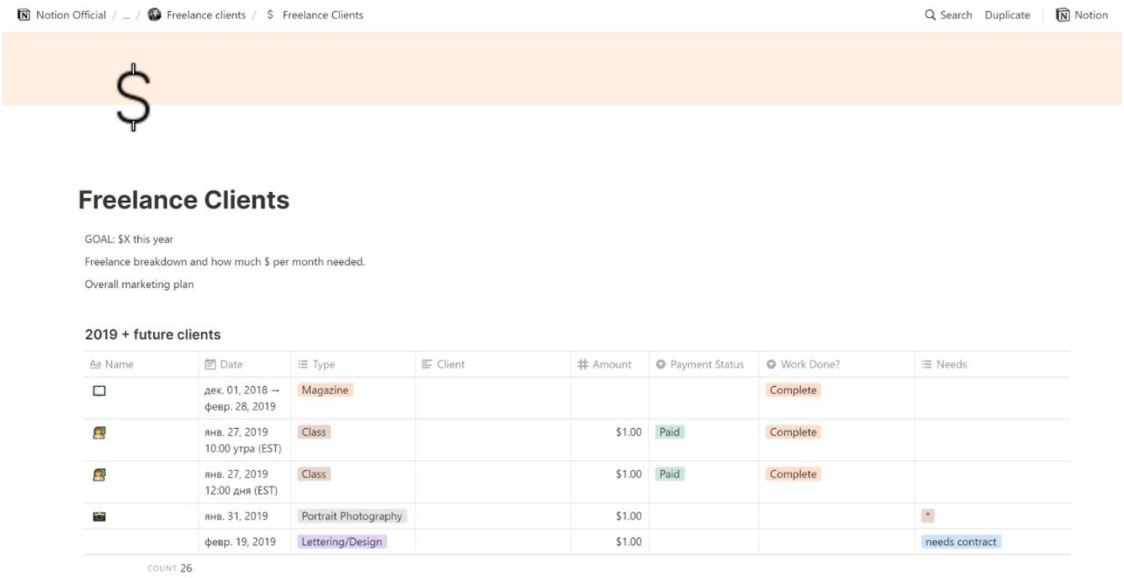

Рисунок 1.3 – Інтерфейс Notion

Серед критеріїв ефективного рішення для підтримки проєктних робіт можна відзначити наступні: простота інтерфейсу, функціонал ведення проєктів, підтримка роботи в команді, функціонал сповіщень користувача та функціонал ведення нотатків.

Таблиця 1.1 – Порівняльний аналіз систем Trello та Notion.

Критерії	Trello	Notion
Функціонал ведення проєктів	+	+
Простота інтерфейсу	+	–
Підтримка роботи в команді	+	+
Функціонал сповіщень користувача	–	+
Можливість ведення нотатків	–	+

Проаналізувавши продукти аналоги представлені на ринку, було визначено головні критерії якими повинна володіти інформаційна технологія проєктування системи для підтримки проєктної діяльності з використанням хмарних сервісів для досягнення успіхів.

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір базових технологій розробки

Сучасний підхід до розробки веб-додатків передбачає ретельний відбір технологій для забезпечення високої продуктивності, надійності та ефективного управління проєктним процесом. Обрання правильного стеку технологій є критично важливим етапом, визначаючим успішність впровадження та подальший розвиток веб-додатку.

Фундаментальним вибором визначаючим подальший шлях у пошуку рішень є вибір мови програмування. Серед великого списку мов для розробки веб-застосунків було обрано Java [8, 9]. Мова програмування Java впродовж десятків років тримає високі рейтинги популярності серед мов для розробки мобільних та веб-додатків. Такою популярністю вона завдячує кросплатформенності та об’єктній орієнтованості, зручність та легкість у використанні привела до розвитку спільноти розробників, які розробили цілу екосистему з бібліотек та фреймворків. Одним з таких фреймворків є фреймворк для розробки веб-додатків на мові Java – Spring [10].

Фреймворк Spring та його модуль Spring Boot [11] є дуже потужним та популярним інструментарієм для розробки веб-орієнтованих систем. Він відомий своєю модульністю, інверсією управління та контейнером введення-виведення, Spring дозволяє створювати гнучкі та ефективні застосунки. Він включає різноманітні модулі, такі як Spring MVC (Model-View-Controller) [12] для веб-розробки, Spring Data [12] для роботи з базами даних, та Spring Security [12] для забезпечення безпеки. Spring Boot є розширенням Spring Framework, спрямованим на полегшення процесу розробки, конфігурації та розгортання додатків. Його концепція "угода головніша за конфігурацію" (convention over configuration) – спрощує налаштування, дозволяючи розробникам швидше перейти до написання бізнес-логіки.

Для реалізації роботи з бізнес-логікою неможливо обійтися без використання баз даних. Тому важливо підібрати доцільну для проєктованої

системи, яка буде задовольняти всі потреби функціональних можливостей інформаційної системи. У розробці веб-додатку було обрано базу даних PostgreSQL [13], яка є однією з найпопулярніших та потужних систем управління базами даних (СУБД) [14], визнаною своєю надійністю, розширеними можливостями та високою продуктивністю. Використання PostgreSQL у веб-додатках дозволяє забезпечити ефективне зберігання, управління та отримання доступу до даних. PostgreSQL відома своєю винятковою стабільністю та надійністю в роботі з великими обсягами даних, долаючи високі навантаження та забезпечуючи довгі періоди безперебійної роботи. Ця система управління базами даних також славиться своїми розширеними можливостями SQL, включаючи складні запити та агрегації, що робить її потужним інструментом для взаємодії з даними. Вбудована підтримка JSON дозволяє зберігати та оптимізовано взаємодіяти з документ-орієнтованими даними, а геопросторові можливості PostgreSQL стають важливим аспектом роботи з географічними даними. Гнучкість та розширюваність дозволяють визначати власні типи даних, функції та операції, надаючи велику гнучкість у роботі з різноманітними видами інформації. Підтримка великої спільноти розробників та користувачів, а також активний розвиток та регулярні випуски оновлень, підтримують PostgreSQL на передовому рівні технологій.

Автоматизація процесу контролю версій бази даних буде реалізовано з використанням інструментарію Flyway [15], що надає можливість синхронізувати стан структури бази даних до актуальної версії застосунку, та змінювати її структуру в залежності від обраної версії програми. Також він підтримує графічний інтерфейс для відстеження зміни в об’єктах бази даних і керування сценаріями міграції в системі керування версіями.

Розробка розподілених систем із великою кількістю мікросервісів часто стикається з викликом забезпечення ефективної комунікації між цими сервісами. Традиційний підхід, який включає в себе ручне написання коду для

кожного клієнта, може призвести до збільшення об'єму коду та складнощі управління його оновленням.

Spring Cloud OpenFeign пропонує елегантне рішення для цієї проблеми, надаючи декларативний підхід до взаємодії між сервісами. Замість того, щоб розробникам вручну налаштовувати кожний клієнтський виклик, OpenFeign дозволяє розробити інтерфейси з анотаціями, що визначають віддалені сервіси та їхні методи.

Це полегшує розуміння та управління кодом, а також зменшує його обсяг, оскільки весь boilerplate-код, пов’язаний з взаємодією, прихований в бібліотеці OpenFeign. Такий підхід сприяє підвищенню продуктивності розробників і полегшує підтримку великої кількості сервісів в архітектурі мікросервісів. Таким чином, Spring Cloud OpenFeign вирішує проблему складності та обсягу коду, необхідного для взаємодії сервісів у розподіленому середовищі.

2.2 Вибір реалізації технології сповіщень

Для реалізації сповіщень користувача проєктованої системи було обрало месенджер Telegram, який відомий своєю широкою популярністю та зручністю використання серед користувачів. Забезпечення сповіщень через даний месенджер дозволяє додатку надсилати повідомлення користувачам безпосередньо на їхні мобільні пристрої чи комп’ютери, що суттєво підвищує гнучкість у використанні розроблюваної інформаційної системи. Для роботи з Telegram у веб-додатках використовується Telegram Bot API [16, 17].

Telegram Bot API є важливим інструментом для взаємодії веб-додатків із Telegram-ботами. За допомогою цього API, розробники можуть створювати чат-ботів та інтегрувати їхню функціональність у свої проєкти. Створення Telegram-бота починається з отримання токена від BotFather, який надає засоби управління та налаштуванням ботів.

API дозволяє відправляти текстові повідомлення, а також мультимедійні файли, спрощуючи комунікацію між веб-додатком та Telegram. Розробники

можуть налаштовувати ботів на відповідь на конкретні команди чи запитання від користувачів, надаючи інтерактивний досвід.

Чат-бот Telegram може працювати у двох режимах Polling та Webhook. Важливо визначити, який саме режим буде обрано при розробці бота для сповіщень. У першому випадку Polling є режимом у якому чат-бот сам проводить опитування сервери Telegram для визначення чи не з’явилось нових оновлень та при визначенні активності в чаті реалізує певний функціонал. Такий підхід створює певну затримку в спілкуванні користувача з додатком та створює постійне навантаження на сервер через регулярне опитування.

У противагу Polling підходу існує Webhook режим, за яким сам Telegram викликає обробник подій якщо користувач створює активність у чаті. Таким чином Webhook режим повністю мінімізує затримки обміну повідомленнями між користувачем та ботом, тому є більш доречною при розробці функціоналу сповіщень для веб-додатку підтримки проєктної діяльності.

Використання інтерактивних клавіатур полегшує взаємодію між користувачем та ботом, а API забезпечує обробку вхідних даних та отримання оновлень. Безпека використання API включає обережне зберігання та обробку токенів для автентифікації. Узагальнюючи, Telegram Bot API робить взаємодію з Telegram простою та ефективною, дозволяючи веб-додаткам реалізовувати різноманітні функції та полегшуючи спілкування з користувачами через цей месенджер.

Також при розробці функціоналу сповіщень буде використано бібліотеку для планування задач Quartz [18]. Ця бібліотека надає потужні можливості для розробки функцій з певним інтервалом та періодичністю. Додатково вона підтримує зручне налаштування через конфігурації Spring та можливість використання Spring Security для забезпечення доступу до ресурсів.

2.3 Вибір функціональних бібліотек

У вимірах сучасної веб-розробки, де кожен вибір технології може суттєво вплинути на продуктивність та ефективність, обрання правильної системи обміну повідомленнями стає ключовою вирішальною точкою. Серед множини доступних технологій вирізняються дві популярні: Apache Kafka [19] і RabbitMQ [20]. Давайте глибше розглянемо їх характеристики, контекст використання та вплив на веб-додатки, щоб зрозуміти, яка система може більш ефективно відповідати вимогам веб-розробки.

Для веб-додатків надійність та ефективність є критичними, вибір між Apache Kafka та RabbitMQ може бути вагомим завданням. Apache Kafka відзначається своєю масштабованістю та здатністю обробляти великі потоки даних в реальному часі. Це робить його ідеальним варіантом для застосунків, де важлива швидкість обміну даними та гарантія їх доставки. З іншого боку, RabbitMQ пропонує зручний та простий підхід до обміну повідомленнями. Його фокус на AMQP протоколі робить його ефективним для використання в архітектурі мікросервісів та сценаріях, де важлива гнучкість взаємодії та легкість впровадження.

Із урахуванням особливостей веб-додатків, де пріоритетом є простота та зручність, а також можливість ефективно впоратися із помірним обсягом даних, RabbitMQ може виявитися більш доцільним вибором. Його прямолінійний підхід і легка інтеграція можуть забезпечити ефективний обмін повідомленнями у веб-додатку, забезпечуючи при цьому високий рівень надійності та простоти використання.

2.4 Додаткові технології розробки

У сучасному підході до веб-розробки Docker [21] та Jenkins [22] стали ключовими інструментами для розробників, спрямованими на поліпшення ефективності, стабільності та прискорення процесів розгортання та постійної інтеграції.

Docker вирізняється якісною ізоляцією та стандартизацією середовищ від розробки до виробництва. Завдяки контейнеризації, усі компоненти

додатку та його залежності упаковуються в контейнери, що дозволяє легко реплікувати однакове середовище для розробки, тестування та виробництва. Це дозволяє уникнути проблем, пов’язаних з різницею у середовищах, та гарантує консистентність усіх етапів розробки.

Jenkins, у свою чергу, впроваджує концепцію постійної інтеграції та постійного розгортання (CI/CD) [22]. Це означає, що Jenkins автоматизує процеси тестування, збірки та розгортання коду. При внесенні змін у репозиторій, Jenkins запускає автоматизовані тести та, у випадку успішного проходження, ініціює процес збірки та розгортання. Це дозволяє розробникам оперативно виявляти та виправляти помилки, а також швидко впроваджувати новий функціонал.

Разом Docker і Jenkins створюють невід'ємну інфраструктуру для автоматизації і раціоналізації розробки. Docker забезпечує стандартизоване середовище для запуску додатку, тим самим гарантуючи його працездатність на різних етапах життєвого циклу, а Jenkins, завдяки CI/CD, забезпечує швидку зміну та розгортання коду. У результаті, ця інтеграція допомагає розробникам зосередитися на творчому процесі, зменшити час відладки та забезпечити стабільні та безпечні релізи.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Проєктування архітектури веб-додатку

Веб-додаток був побудований на основі мікросервісної архітектури. Архітектура додатку зображена на рисунку 3.1.

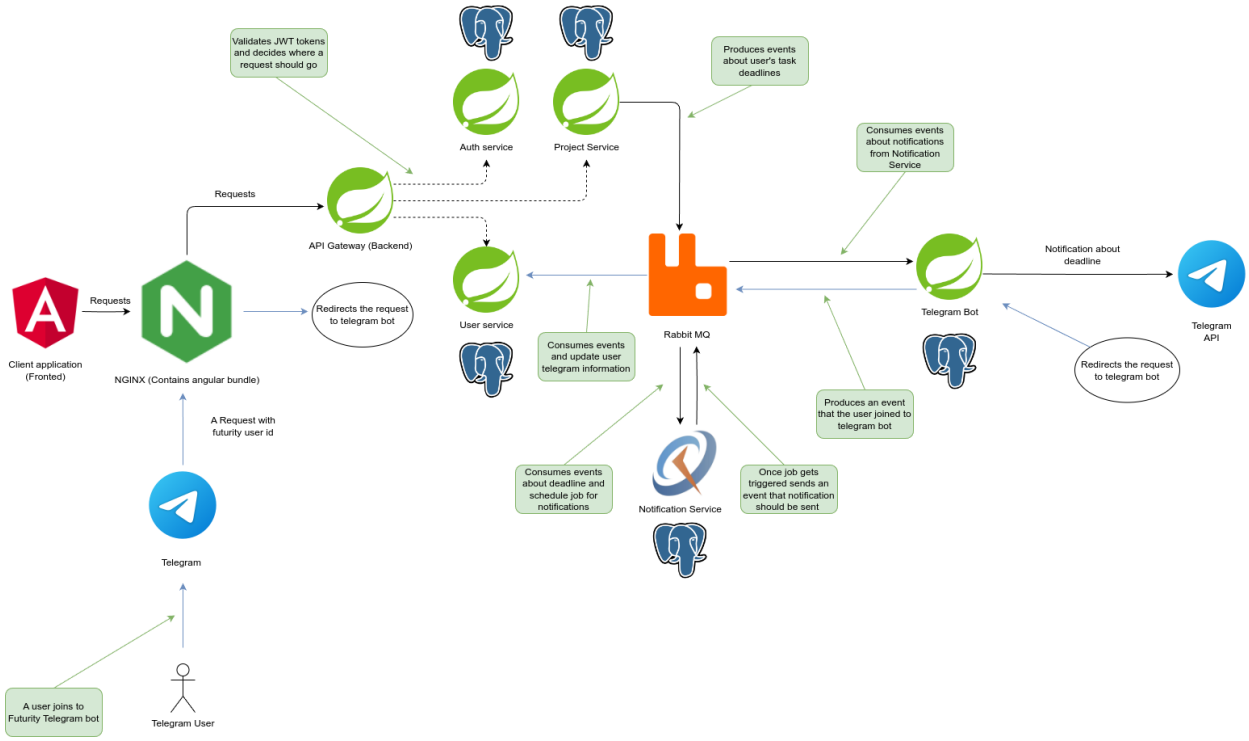

Рисунок 3.1 – Архітектура веб-додатку

Веб-додаток ґрунтується на клієнт-серверній архітектурі, яка включає два основних елементи: клієнта та сервера. Клієнт – це комп’ютер користувача, що висилає запити серверу для отримання інформації або виконання операцій. Сервер – це комп’ютер, який обслуговує запити клієнтів, надає доступ до ресурсів та зберігає інформацію та бази даних. У такій системі клієнт висилає запит серверу, який обробляє його та висилає результат користувачеві. Сервер може обслуговувати декілька клієнтів одночасно, утримуючи запити в черзі та виконуючи їх послідовно при одночасному надходженні. Інколи запити мають пріоритети, і ті з вищим пріоритетом обробляються першими.

Backend частина складається з 6 мікросервісів, кожен з яких виконує певну свою частину роботи. Мікросервіс "API Gateway" реалізує популярний паттерн у мікросервісній архітектурі, орієнтований на розділення клієнтського програмного інтерфейсу від внутрішньої реалізації та виконання аутентифікації. Цей мікросервіс обробляє всі запити як зворотний проксі, витягуючи ресурси від внутрішніх мікросервісів від імені клієнтської програми. Мікросервіс "Auth service" відповідає за авторизацію, включаючи функції логіну, реєстрації і розсилки повідомлень на електронну пошту. Мікросервіс "User service" відповідає за управління користувачами та їхніми даними в системі, включаючи збереження, додавання, пошук і редагування інформації про користувачів (логін, пошта, аватар і інше). “Project service” відповідальний за керування проєктами та задачами користувача, та обробку їх інформації (опис, термін виконання, пріоритет, назви і інше). Мікросервіс “Notification service” відповідальний за обробку термінів виконання задачі, тобто їх налаштування та зберігання, також він відсилає нотифікації, коли наступає термін виконання задачі. Цей мікросервіс використовує Quartz фреймворк щоб планувати терміни та підтримки персистентності, тобто терміни та їх стани зберігаються в базі даних, що дозволяє відновлення планування після перезапуску мікросервісу. Мікросервіс “Telegram Bot” відповідальний за інтеграцію з Telegram, він керує даними користувачів, які під’єднали акаунт Telegram до веб-додатку, також приймає та оброблює запити від Telegram (тобто від користувачів), також формує та надсилає повідомлення про терміни користувачам в Telegram.

Кожний мікросервіс має свою власну базу даних, таким чином реалізовуючи патерн в мікросервісної архітектури “Database-per-Service” (База даних на сервіс). Цей підхід передбачає, що дані кожного мікросервісу зберігаються в окремій базі даних, що є незалежною від інших мікросервісів. Такий підхід дозволяє ізолювати дані, зменшує взаємозалежність та сприяє незалежному розвитку кожного сервісу. RabbitMQ виступає як “Message Broker”, та за допомогою нього реалізується патерн “Publish/Subscribe”

(видавець/підписник), який є поведінковим патерном проєктування, де об'єкт, відомий як видавець, передає повідомлення (події або зміни) декільком об'єктам-підписникам, які виражають інтерес до цих подій. При цьому видавець не має жорсткої прив'язки до конкретних підписників, що дозволяє легко розширювати та модифікувати систему.

RabbitMQ використовується для комунікації між мікросервісами та передачі інформації про терміни задач, наприклад, коли користувач створив задачу, або коли термін задачі наступив, і треба зробити якісь дії. Також він передає інформацію про Telegram, наприклад, коли користувач доєднався до додатку. Використання RabbitMQ дозволяє ефективно реалізувати розподілені та взаємодіючі системи, де об'єкти можуть реагувати на зміни в інших частинах системи без прямого зв'язку між ними.

Застосування такого підходу допомагає забезпечити слабку зв’язаність між об'єктами та сприяє розширюваності системи, забезпечуючи ефективний механізм для реакції на події та сповіщення зацікавлених сторін.

Для забезпечення доступу до захищених ресурсів використовується JWT токен, який генерується мікросервісом авторизації та аутентифікації за допомогою приватного ключа. “Api-Gateway” мікросервіс перевіряє цей токен за допомогою публічного ключа. Принцип REST [23] – незалежність від стану (stateless), що вимагає від клієнта самостійно проводити аутентифікацію при кожному запиті. Для цього використовується JWT токен, який унікально ідентифікує користувача. Токен видається при логіні і передається при кожному наступному запиті, щоб сервер міг його перевірити та авторизувати користувача. Для підвищення безпеки токен може мати обмежений термін дії, після якого він стає недійсним.

Frontend частина побудована на SPA (Single-Page-Application) [24], тобто вона завантажується один раз, а весь контент динамічно оновлюється без перезавантаження сторінки. У SPA весь інтерфейс відображається на одній сторінці, що дозволяє користувачам плавно взаємодіяти з додатком, зменшуючи час завантаження та забезпечуючи більш зручний та

інтерактивний досвід використання веб-додатка. Frontend використовує підхід Client-Side Rendering (CSR), це підхід, при якому весь процес відображення веб-сторінки відбувається на боці клієнта, а не на сервері. При CSR веб-додаток завантажує основний HTML-код на клієнтський браузер, а потім використовує JavaScript для динамічного завантаження та відображення контенту без повторного запиту до сервера. Цей підхід дозволяє створювати більш інтерактивні та швидкодіючі веб-додатки, оскільки зменшує час очікування завантаження сторінки.

Перший хто «зустрічає» запит від клієнту – це Nginx. Його основна функціональність включає обробку статичного та динамічного контенту, балансування навантаження та виконання ролі реверсивного проксі. У ролі реверсивного проксі, Nginx приймає запити від клієнтів і пересилає їх до внутрішніх серверів, де обробляються запити та повертається відповідь клієнтові. Це дозволяє розподіляти навантаження між різними серверами, підвищує надійність та забезпечує більш ефективне використання ресурсів. В якості веб-сервера, Nginx ефективно обробляє статичний контент, такий як HTML, CSS та зображення, а також може взаємодіяти з додатками чи серверами, які відповідають за динамічний контент. Його висока продуктивність та низький використання ресурсів роблять його популярним вибором для розгортання веб-додатків та обслуговування великої кількості одночасних з’єднань.

3.2 Налаштування середовища розробки

Автоматизація деяких процесів розробки може значно вплинути на швидкість та ефективність всього процесу роботи над проєктом. Тому для полегшення розробки було інтегровано CI/CD систему Jenkins. Jenkins надає широку підтримку кастомізації за допомогою різних плагінів, також надає зручний інтерфейс для керування Jobs та Pipelines. Інтерфейс Jenkins зображено на рисунках 3.2-3.3.

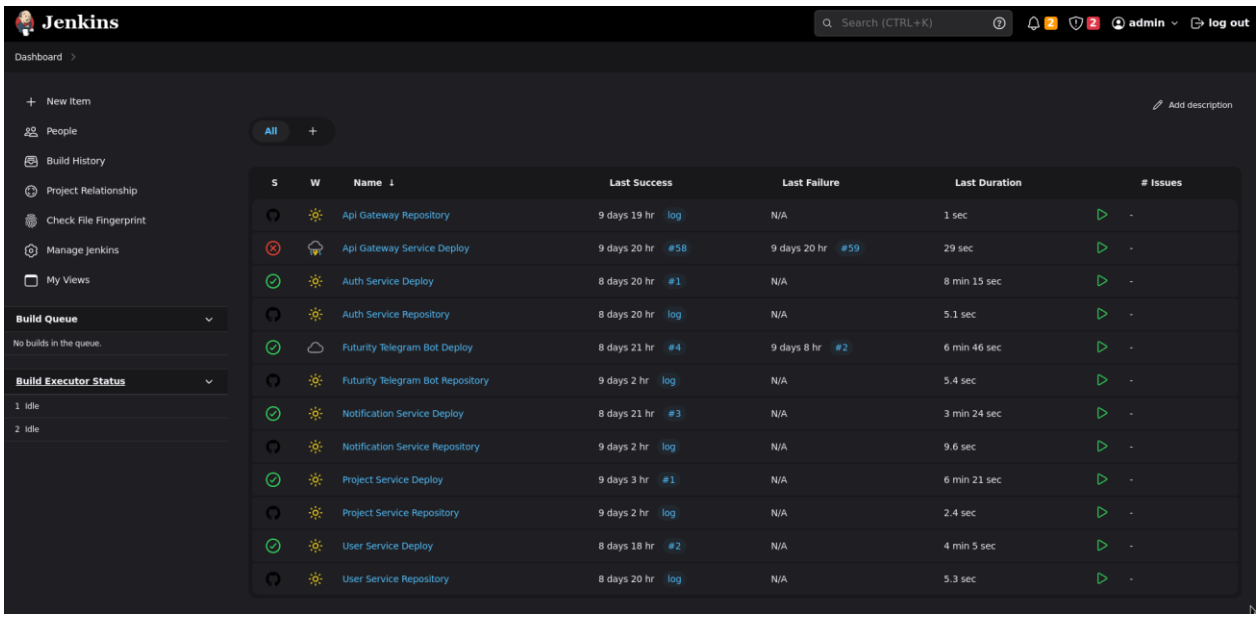

Рисунок 3.2 – Головна сторінка Jenkins (Dashboard)

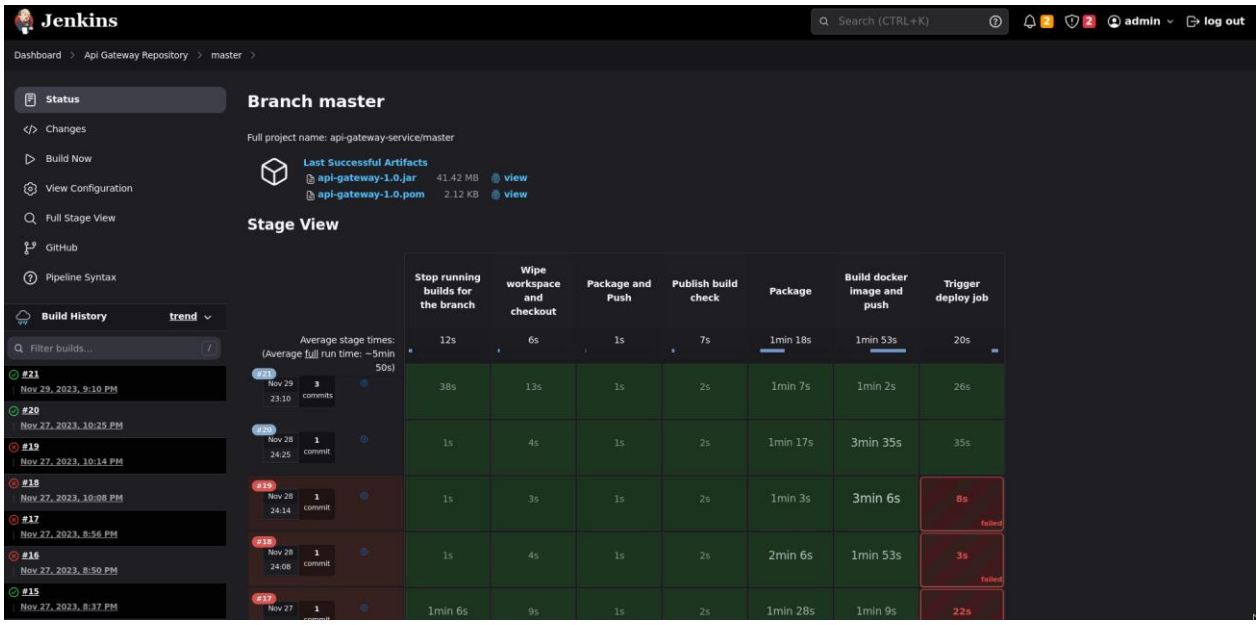

Рисунок 3.3 – Сторінка Job збірки проєкту

Для реалізації CI/CD Pipeline було використано такі інструменти як Docker, щоб забезпечити легку і швидку контейнеризацію програмного забезпечення та спростити розгортання мікросервісів, полегшивши керування їх залежностями та конфігурацією. Docker-compos для управління багатоконтейнерним середовищем, , що полегшує організацію та розгортання веб-додатку. Схема розробленого CI/CD Pipeline зображена на рисунку 3.4.

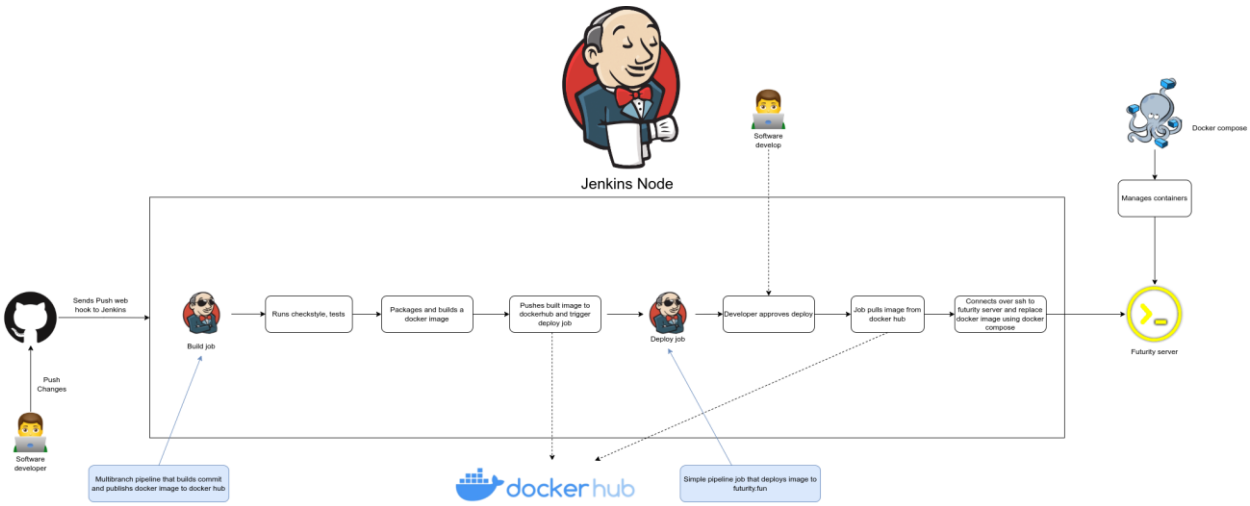

Рисунок 3.4 - CI/CD Pipeline

CI/CD Pipeline має 2 головних етапа:

– збірка та перевірка коду;

– розгортання нового коду на віддаленому сервері.

Як показано на рисунку 3.2, перший етап – це відправка нового коду в систему контролю коду, у даному випадку Github. Він далі вже надсилає WebHook, тобто повідомлення Jenkins серверу, що новий код був завантажений. Після цього Jenkins запускає першу Job - Build commit (Збірка коміту). До цього процесу входять такі етапи:

– компіляція;

– тестування;

– аналіз коду на якість (Checkstyle, Code smells);

– збірка Docker образу та його публікацію на DockerHub;

– виклик нової Job на розгортання нової версії.

Також можна виділити синхронізацію Jenkins серверу з платформою Github. Розробник прямо сторінці проєкту може переглянути статус збірки його нового коду, на рисунка 3.5-3.6 представлено приклади вдалої та невдалої збірки.

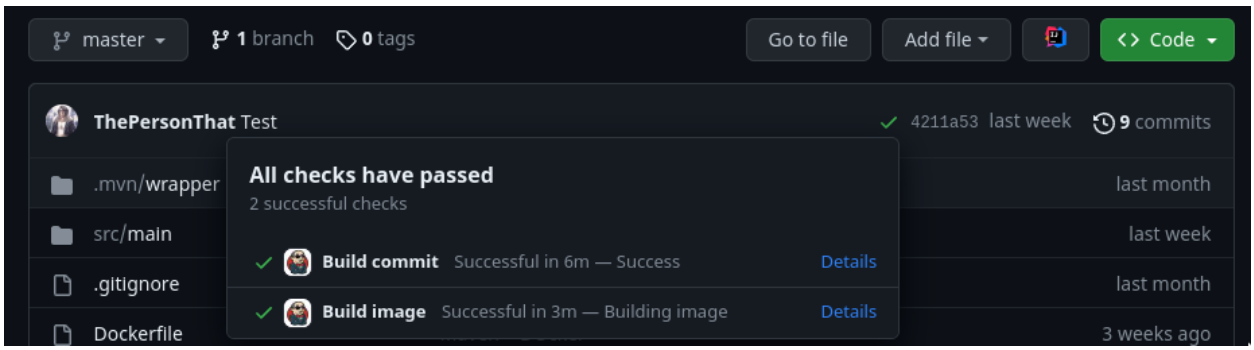

Рисунок 3.5 – Приклад вдалої збірки

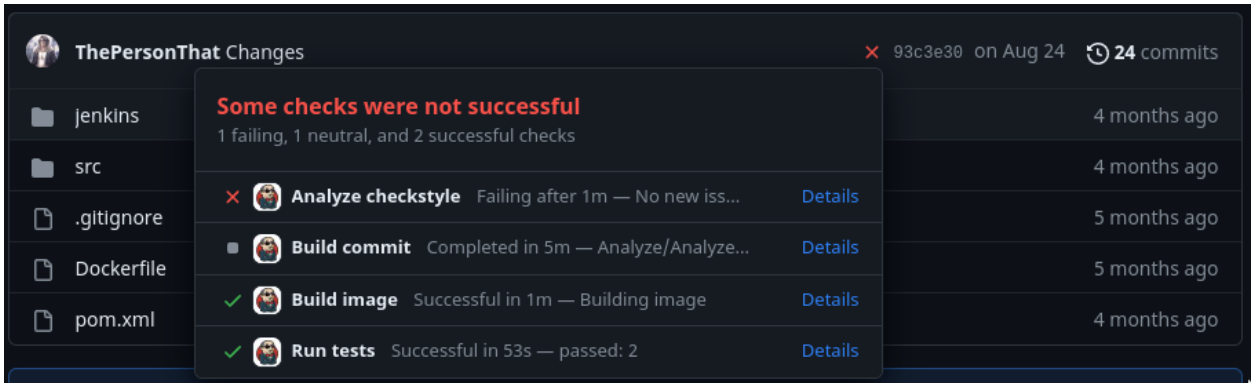

Рисунок 3.6 – Приклад не вдалої збірки

Після того як Job на розгортання була викликана, вона буде чекати підтвердження від розробника, чи розгортати нову версію коду на віддаленному сервері чи ні. Якщо розробник підтвердив розгортання нової версії, то Jenkins підключиться до віддаленного серверу по SSH, що дозволить йому виконувати команди на віддаленному сервері, та розгорне новий Docker образ, який завантажить з DockerHub, за допомогою Docker Compose, а стару версію коду (тобто якогось одного мікросервісу) він зупинить та видалить.

Використання CI/CD pipeline допомогло значно скоротити час на розробку та розгортання нових версій веб-додатку.

3.3 Проєктування бази даних

Після створення архітектури слід спроєктувати базу даних, використовуючи систему управління базами даних. Результат проєктування включає визначену структуру з таблиць та логічних зв’язків. Визначення

структури таблиці включає типи даних, розмір та послідовність стовпців, а також первинний ключ.

Перший етап проєктування бази даних – виділення сутностей, об’єктів предметної області, які зберігаються в базі даних і представлені окремими таблицями. Кожен стовпець містить атрибут, а рядок – екземпляр сутності, що спрощує додавання та редагування інформації на веб-ресурсі.

У процесі проєктування бази даних було визначено наступні сутності:

− Користувач (User) – зберігає дані про користувача;

− Код підтвердження (Confirmation token) – сутність для верифікації електронної пошти користувача;

− Refresh токен – генерація нових токенів доступу;

− Проєкт (Project) – зберігає дані проєкту;

− Колонка проєкту (Project Column) – зберігає дані про колонку проєкту;

− Завдання (Task) – сутність яка зберігає дані про задачу;

− Користувач телеграму (Telegram_user) – зберігає дані про обліковий запис Telegram.

Опис полів для сутності «User» наведено в таблиці 3.1.

Таблиця 3.1 – Опис полів сутності «User»

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Ідентифікація користувача
email	varchar(100)	Not null, Unique	Електронна пошта користувача
nickname	varchar(100)	Not null	Ім’я користувача
password	varchar(100)	Not null	Пароль користувача
avatar	blob	Not null	Аватар користувача
has_telegram	boolean	Not null	Наявність Telegram у користувача

Опис полів для сутності «Confirmation token» наведено в таблиці 3.2.

Таблиця 3.2 – Опис полів сутності «Confirmation token»

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Ідентифікація коду підтвердження
code	char(100)	Not null, Unique	Код підтвердження
confirmed	boolean	Not null	Чи пройшла пошта верифікацію
confirmed_at	timestamp	Nulll	Дата, коли була підтверджена пошта користувача
expired_at	timestamp	Not Null	Дата, коли код підтвердження стане не дійсним

Опис полів для сутності «Refresh» наведено в таблиці 3.3.

Таблиця 3.3 – Опис полів сутності “Refresh токен”.

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Ідентифікація refresh токену
refresh_token	varchar(400)	Not null, Unique	Використується для генерування нових access токенів

Опис полів для сутності “Project” наведено в таблиці 3.4.

Таблиця 3.4 – Опис полів сутності “Project”.

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Ідентифікація проєкту
user_id	bigint	Not null	Використовується для зв’язку з сутністю “Користувач” до якого відноситься проєкт
name	varchar(60)	Not null	Назва проєкту
description	varchar(1000)	Not null	Опис проєкту
preview	blob	Not null	Прев’ю проєкту

Опис полів для сутності «Project Column» наведено в таблиці 3.5.

Таблиця 3.5 – Опис полів сутності «Project Column»

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Ідентифікація колонки проєкту
index	integer	Not null	Індекс колонки по відношенню до інших колонок в проєкті
name	varchar(60)	Not null	Назва колонки

project_id	bigint	Not null	Використовується для зв’язку з сутністю «Проєкт» до якої відноситься колонка
preview	blob	Not null	Прев’ю проєкту
done_column	boolean	Not null	Використовується для з’ясування чи колонка для виконаних задач чи ні.

Опис полів для сутності «Task» наведено в таблиці 3.6.

Таблиця 3.6 – Опис полів сутності «Task»

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Використовується для ідентифікації задачі
index	integer	Not null	Індекс задачі по відношенню до інших колонок в проєкті
name	varchar(60)	Not null	Назва задачі
column_id	bigint	Not null	Використовується для зв’язку з сутністю “Колонка проєкту” до якої відноситься задача
deadline	timestamp	Not null	Термін виконання задачі
is_completed	boolean	Not null	Стан виконання задачі

Опис полів для сутності «telegram_user» наведено в таблиці 3.6.

Таблиця 3.6 – Опис полів сутності «telegram_user»

Назва	Тип	Обмеження	Опис
id	bigint	Primary Key	Ідентифікація облікового запису Telegram
chat_id	bigint	Not null	Ідентифікація чату в Telegram
Futurity_user_id	bigint	Not null	Ідентифікація облікового запису в веб-додатку

Модель бази даних веб-додатку для підтримки проєктної діяльності з використанням хмарних сервісів зображено на рисунку 3.7.

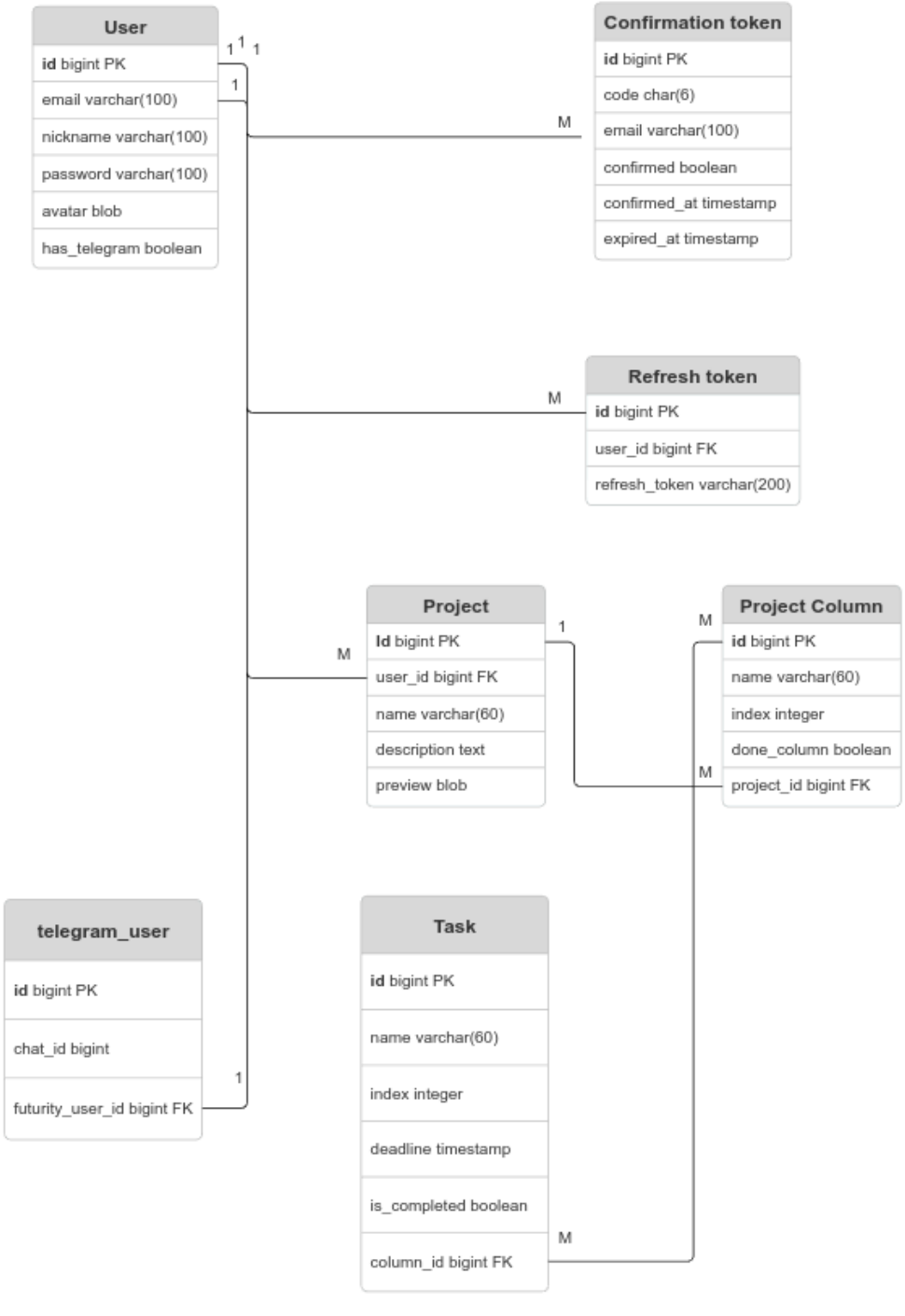

Рисунок 3.7 – Модель бази даних веб-додатку

3.4 Програмна реалізація системи сповіщень

Для реалізації функціоналу сповіщень було обрано технологію Telegram Bot API. В свою чергу Telegram Bot API є надбудовою поверх Telegram API яка відкрита та надає можливість спілкуватись з серверами Telegram через протокол MTProto.

Для написання бота працюючого з Telegram Bot API було обрано відкриту Java бібліотеку – TelegramBots.

Так як це найголовніша функціональність веб-додатку, тому треба розглянути більш детально її реалізацію, роботу та взаємодію з Telegram. Спочатку користувач “прив’язує” свій телеграм акаунт до веб-додатку, схема на рисунку 3.8 відображає цей процес.

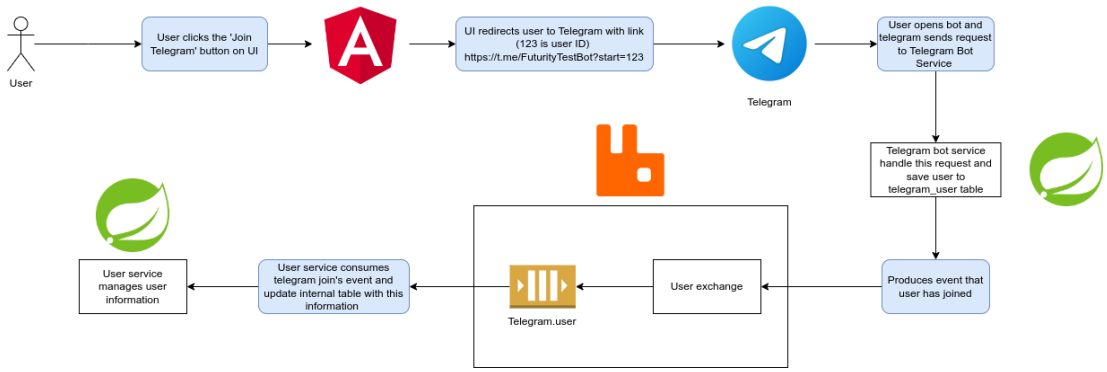

Рисунок 3.8 – Схема роботи системи сповіщень

Якщо розглянути схему більш детально, то можна виділити наступні етапи:

– Користувач на сторінці веб-додатку натискає кнопку “Приєднатись до Telegram”, веб-сторінка переправляє користувача на Telegram бота, і передає до Telegram ID користувача;

– Користувач відкриває Telegram, та натискає кнопку “Start”, і тим самим Telegram відправляє сповіщення до веб-додатку, що користувач доєднався до Telegram бота;

– Telegram Bot Service веб-додатку оброблює цей запит від Telegram, та зберігає інформацію про Telegram користувача, тобто його CHAT_ID та додаткову інформацію в базі даних;

– Далі Telegram Bot Service генерує повідомлення для інших мікросервісів, що користувач доєднався до Telegram. В результаті кожний мікросервіс виконує якісь дії з цією інформацією, наприклад, user-service помічає в себе в базі даних, що даний користувач має під’єднаний Telegram.

Далі розглянемо реалізацію функціоналу відправки сповіщень, схему якої зображено на рисунку 3.9.

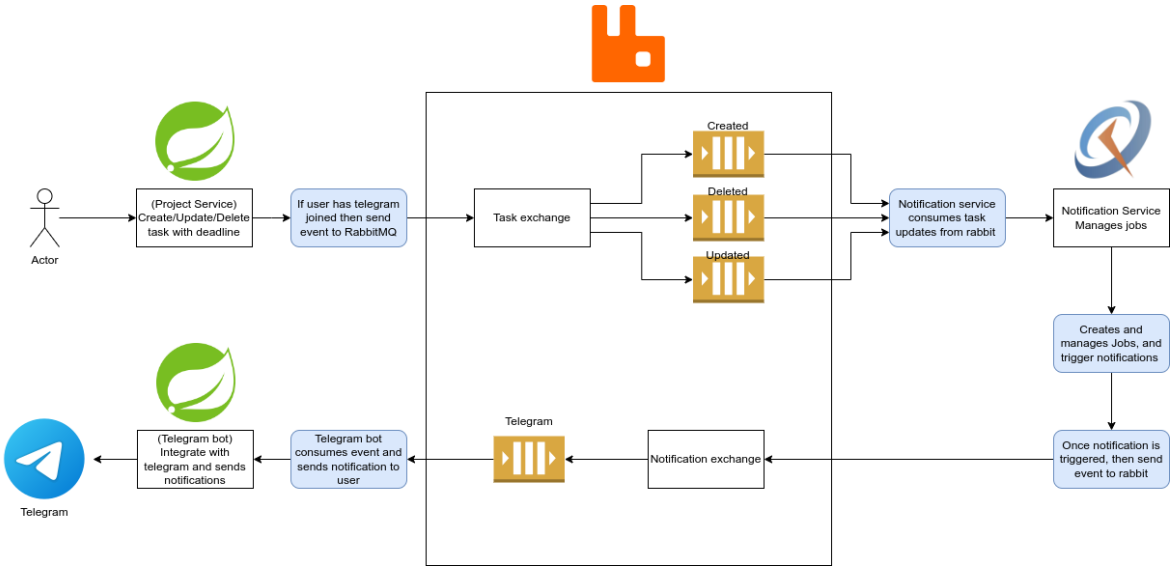

Рисунок 3.9 – Схема реалізації функціоналу відправки сповіщень

Функціонал відправки сповіщень поділяється на наступні етапи:

– Користувач на сторінці веб-додатку створює, видаляє, або редагує задачу, яка має термін;

– Якщо користувач заздалегідь доєднався до Telegram бота веб-додатку, тоді project-service повідомляє інші мікросервіси за допомогою Message Broker (а саме через Task Exachnge), що користувач виконав певну дію з задачею;

– Notification-service отримує це повідомлення та планує тригер з терміном виконання цієї задачі;

– Коли термін виконання задачі наступає, Notification-service повідомляє про це інші сервіси за допомогою Message Broker (а саме через Notificatoin Exachnge), що термін задачі наступив.

– Telegram Bot Service реагує на цю подію та формує повідомлення користувачу та відправляє його до Telegram.

3.5 Налаштування віддаленого серверу

Для надійної роботи системи сповіщень Telegram бота та весь веб-додаток необхідно встановити на віддалений сервер. Хостинг на віддаленому сервері гарантує системі стабільне живлення та інтернет з'єднання.

Орендування хостингу надає білу IP адресу, яка необхідна для реалізації глобального доступу необхідного для підключення до розроблюваного веб-додатку. У процесі роботи було орендовано VPS (Virtual Private Server) [25] який є частиною одного реального сервера розбитого на декілька віртуальних. Сторінка адміністратора віртуального серверу зображена на рисунку 3.10.

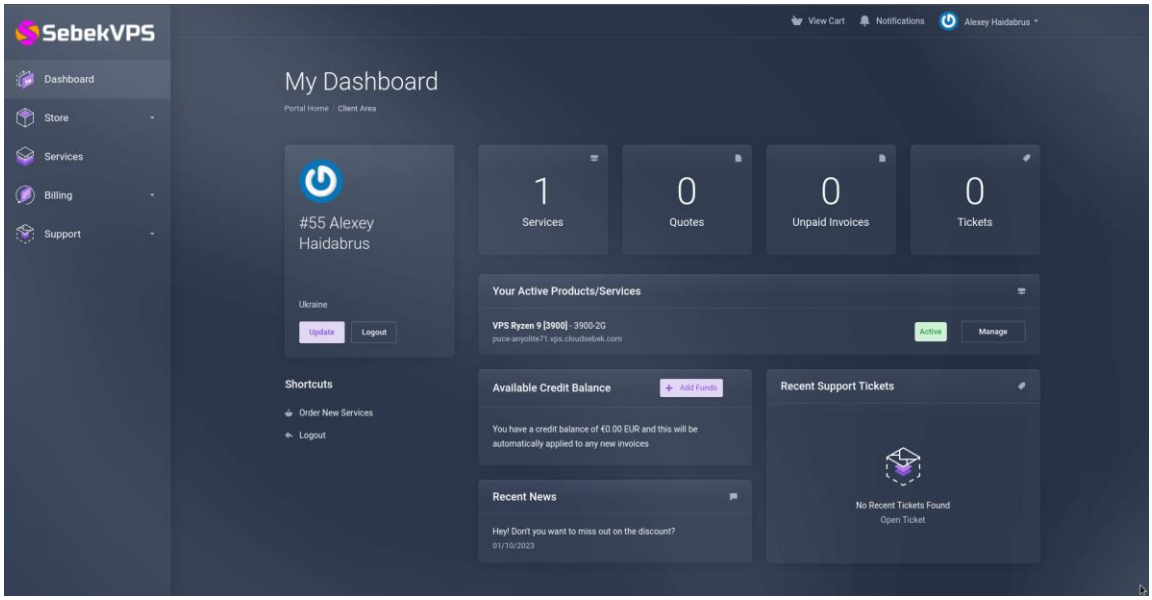

Рисунок 3.10 – Адмін-панель віртуального серверу

Враховуючи важливість гарантування безпеки при обміні даними між клієнтом та сервером необхідно виконати налаштування SSL (Secure Sockets Layer) [26] сертифікатів. Розроблюваний веб-додаток буде використовувати захищений протокол обміну даними між клієнтом та сервером HTTPS. Основними перевагами в використанні HTTPS протоколу є: шифрування даних, підтвердження ідентичності та захист від атак або фішингу. На рисунку 3.11 зображено SSL сертифікат веб-додатку.

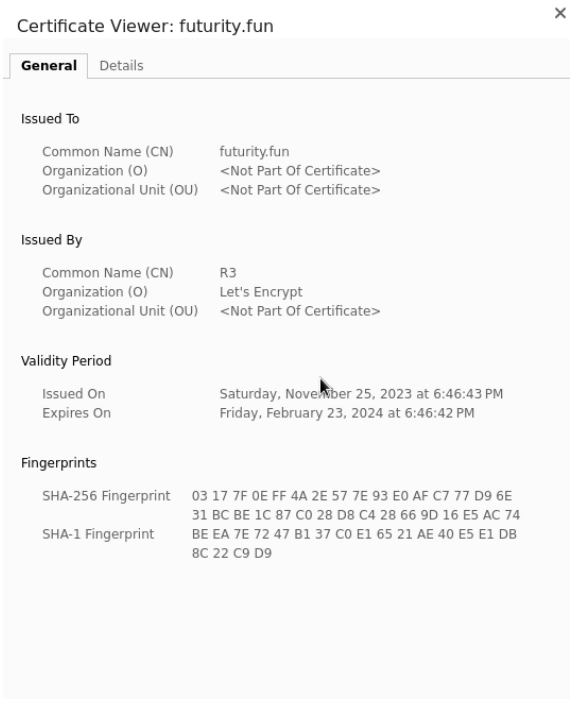

Рисунок 3.11 – SSL сертифікат веб-додатку

Для зручності користувачів адреса веб-додатку повинна складатися з текстового посилання замість цифрової адреси серверу, тому було орендовано відповідну доменну адресу, що зображено на рисунку 3.12.

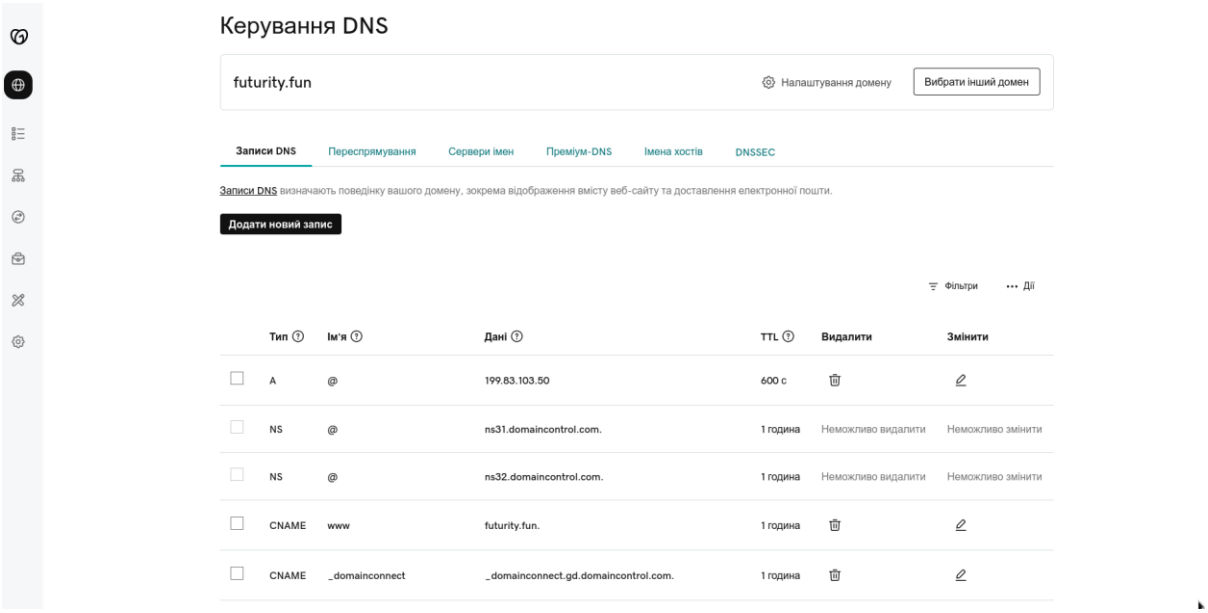

Рисунок 3.12 – Орендування доменної адреси для серверу.

3.6 Тестування реалізованого функціоналу

Функціонал сповіщень з використанням Telegram Bot API, було успішно інтегровано в веб-додаток для підтримки проєктних робіт.

Після реєстрації в веб додатку користувач може доєднати Telegram бота до свого акаунту в налаштуваннях облікового запису, інтерфейс до та після додавання бота зображено на рисунках 3.13 – 3.14.

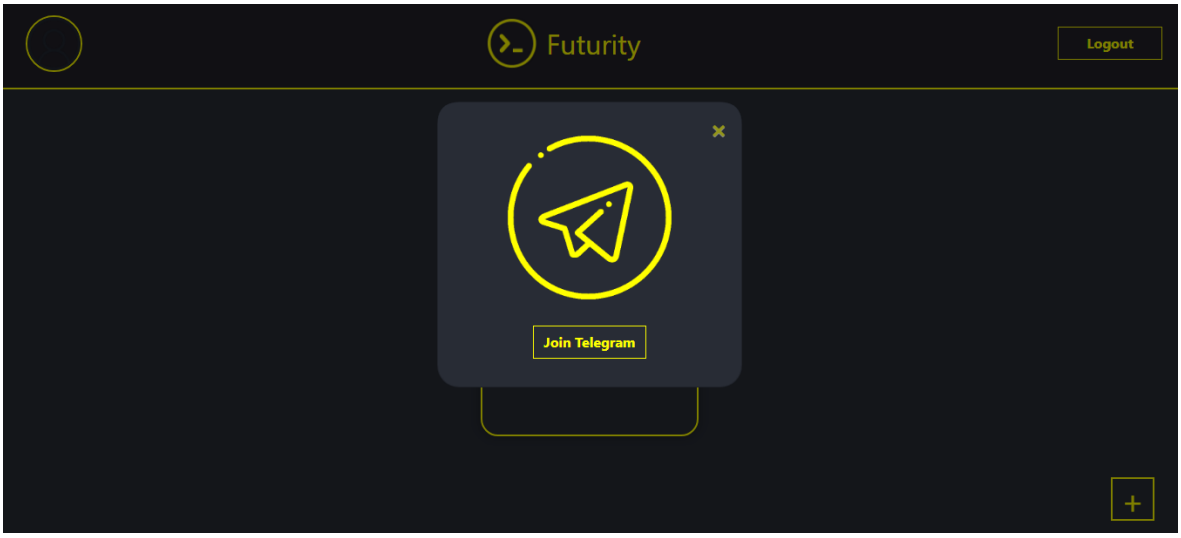

Рисунок 3.13 – Додавання Telegram бота до акаунту

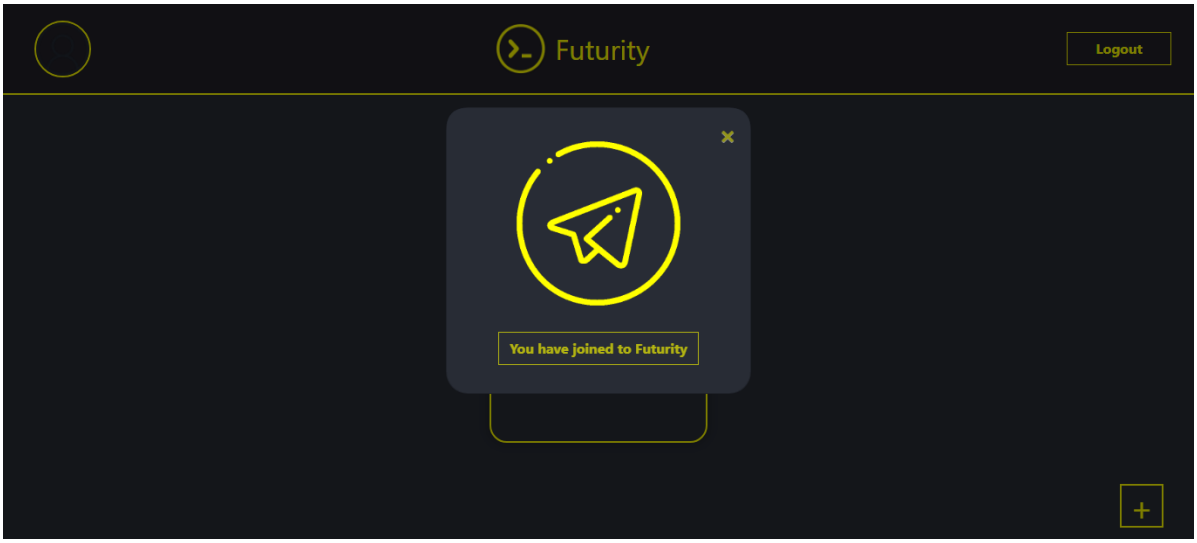

Рисунок 3.14 – Після успішного додавання бота

При переходжені за посиланням Telegram бот – «Futurity Bot» з’явиться у списку чатів та привітає про успішне додавання бота, рисунок 3.15.

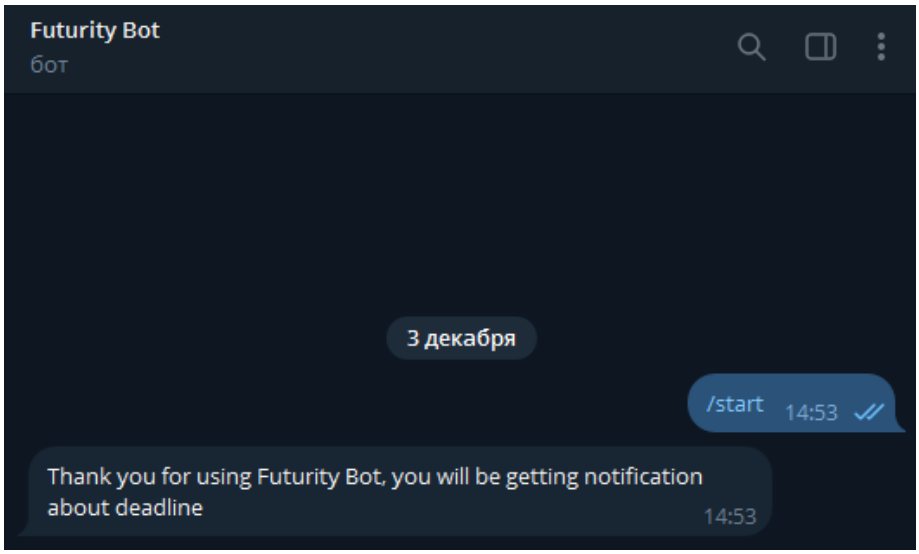

Рисунок 3.15 – Інформування бота про початок роботи

Незалежно від того додав користувач Telegram акаунт чи ні, він може використовувати функціонал Kanban дошки у звичайному режимі. Розроблювати проєкти, визначати плани та задачі, змінювати їх місцями чи позначати виконаними, як показано на рисунку 3.16.

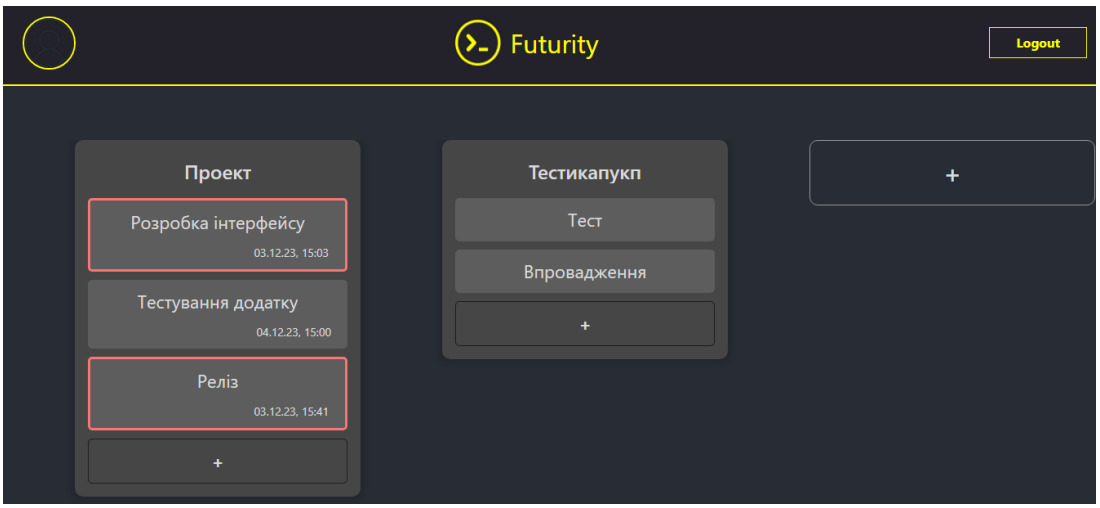

Рисунок 3.16 – Приклад роботи з Kanban дошкою у веб-додатку

Головною особливістю реалізованого функціоналу сповіщень є автоматичне інформування користувача при додаванні Telegram бота до облікового запису, рисунок 3.17.

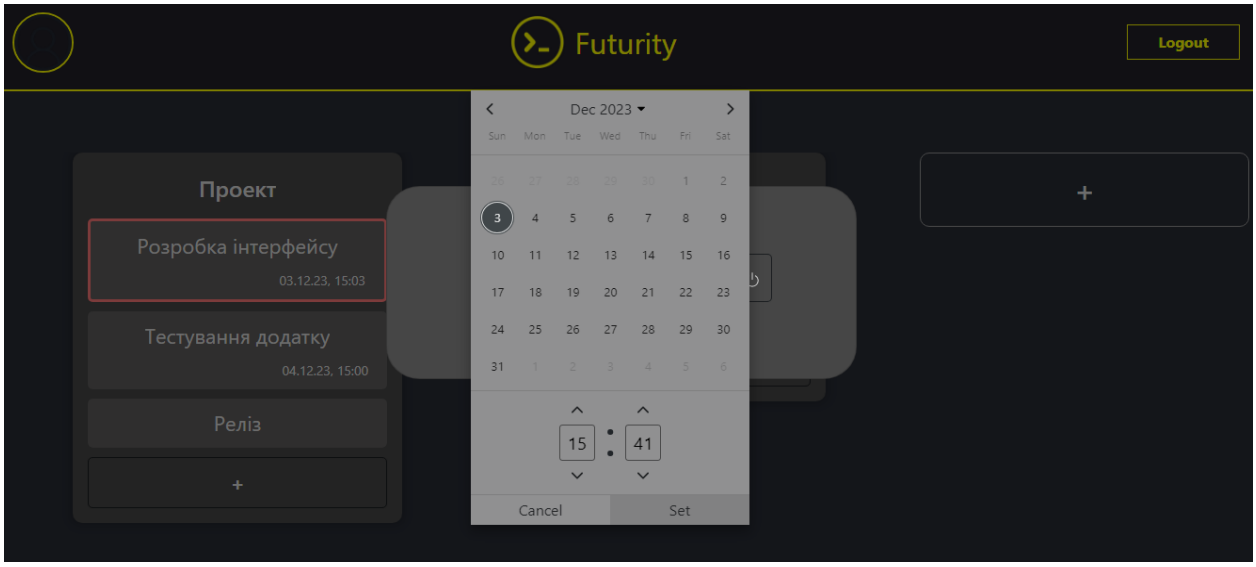

Рисунок 3.17 – Встановлення термінів виконання задачі

Після встановлення терміну виконання задачі сповіщення бот буде інформувати користувача два рази: за один день до визначеного терміну та за 5 хвилин, у випадку якщо користувач не відмітить задачу як виконану раніше.

Як показано на рисунку 3.18 бот інформує користувача скільки часу залишилось до закінчення терміну та назву самої задачі з посиланням на неї.

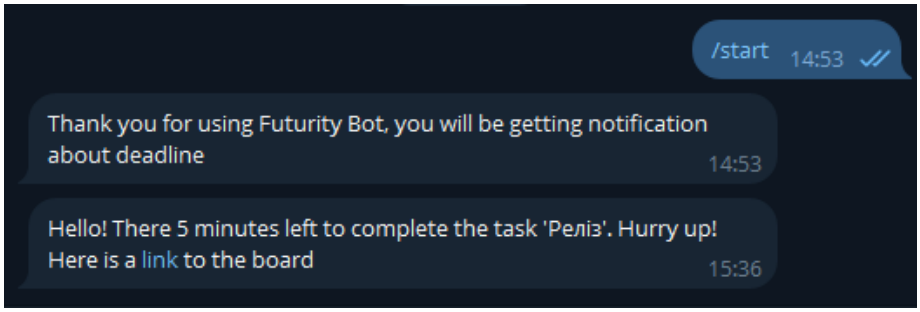

Рисунок 3.18 – Сповіщення про закінчення терміну виконання задачі

Розроблений функціонал сповіщення в поєднанні з функціоналом ведення Kanban дошки спрощує процес планування та ведення проєктних

робіт, адже виключає потребу в додатковому використанні календарів чи інших систем планування та відстеження часу на виконання задач.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи магістра було проведено аналітичні роботи та практичну реалізацію інформаційної системи для підтримки проєктної діяльності з використанням хмарних сервісів.

У процесі аналізу предметної області було проаналізовано основні методики для підтримки проєктної діяльності, визначено цілі та задачі проєкту та проведено аналіз аналогів, визначено їх переваги та недоліки.

У ході аналізу технологій розробки було обрано мову програмування, фреймворк та додаткові бібліотеки для реалізації функціоналу веб-додатку, а також проаналізовано доцільність використання допоміжних інструментів розробника таких як Docker та Jenkins.

У практичній частині кваліфікаційної роботи спроєктовано архітектуру веб-додатку, розроблено базу даних та веб-додатку та його функціонал. Розроблений веб-додаток підтримки проєктної діяльності було успішно розгорнуто на віддаленому приватно сервісі, налаштовано протоколи безпеки та доменне ім’я сайту. В кінці було протестовано правильність роботи функціоналу готового веб-додатку та роботу його системи сповіщень.

Використання розробленої інформаційної системи для підтримки проєктної діяльності з використанням хмарних сервісів сприятиме підвищенню якості ведення проєктних робіт в широкому спектрі галузей.

Лістинг основних модулів розробленого web-додатку представлено у додатку А.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Verwijs, C., Russo, D. A Theory of Scrum Team Effectiveness. ACM Transactions on Software Engineering and Methodology. 2023. Vol. 32, No. 3.

2. Alami, A., Krancher, O. How Scrum adds value to achieving software quality? Empirical Software Engineering. 2022. Vol. 27, No. 7.

3. Gaete, J., Villarroel, R., Figueroa, I., та ін. Agile application approach with Scrum, Lean and Kanban. Ingeniare. 2021. Vol. 29, No. 1.

4. Trebuna, P., Pekarcikova, M., Kliment, M., та ін. Online e-kanban system implementation in a manufacturing company. International Journal of Simulation Modelling. 2023. Vol. 22, No. 1.

5. Солнцев, С. О., Жигалкевич, Ж. М., Залуцький, Р. О. Тенденції розвитку цифрового маркетингу. Journal of Strategic Economic Research. 2023. No. 6.

6. Johnson, H. A. Trello. Journal of the Medical Library Association. 2017. Vol. 105, No. 2.

7. Notion: [Електронний ресурс] - Режим доступу до ресурсу: www.notion.so/product.

8. Ogihara, M. Fundamentals of Java Programming: *Fundamentals of Java Programming*. 2018.

9. Christopher, L., Waworuntu, A. Java Programming Language Learning Application Based on Octalysis Gamification Framework. IJNMT (International Journal of New Media Technology). 2021. Vol. 8, No. 1.

10. Zhang, F., Sun, G., Zheng, B., та ін. Design and implementation of energy management system based on spring boot framework. Information (Switzerland). 2021. Vol. 12, No. 11.

11. Duarte, A. Spring Boot: Practical Vaadin. 2021.

12. Varanasi, B., Bartkov, M. Spring REST: Building Java Microservices and Cloud Applications: *Spring REST: Building Java Microservices and Cloud Applications*. 2021.

13. Juba, S., Vannahme, A., Volkov, A. Learning PostgreSQL: *Pack*

Publishing. 2015.

14. Klimek, B., Skublewska-Paszkowska, M. Comparison of the performance of relational databases PostgreSQL and MySQL for desktop application. Journal of Computer Sciences Institute. 2021. Vol. 18.

15. Flyway Documentation: [Електронний ресурс] - Режим доступу до ресурсу: https://documentation.red-gate.com/flyway.

16. Abd Halim, I. H., Mahamad, A. I., Mohd Fuzi, M. F. Automated Alert System for River Water Level and Water Quality Assessment using Telegram Bot API. Journal of Computing Research and Innovation. 2021. Vol. 6, No. 3.

17. Santoso, W., Nurjannah, W., Shudhuashar, M., та ін. The Development of Telegram Bot Api to Maximize The Dissemination Process of Islamic Knowledge in 4.0 Era. Jurnal teknik informatika. 2022. Vol. 15, No. 1.

18. Scheduling in Spring with Quartz: [Електронний ресурс] - Режим доступу до ресурсу: https://www.baeldung.com/spring-quartz-schedule.

19. Kumar, M., Singh, C. Building Data Streaming Applications with Apache Kafka: *Packt*. 2017.

20. RabbitMQ. RabbitMQ: rabbitmq.com.

21. Anderson, C. Docker / 2015.

22. Dingare, P. P. CI/CD Pipeline Using Jenkins Unleashed: Solutions While Setting Up CI/CD Processes: *CI/CD Pipeline Using Jenkins Unleashed: Solutions While Setting Up CI/CD Processes*. 2022.

23. What is a RESTful API? [Електронний ресурс] - Режим доступу до ресурсу: https://aws.amazon.com/what-is/restful-api/?nc1=h_ls.

24. Fink, G., Flatow, I. Pro Single Page Application Development: *Pro Single Page Application Development*. 2014.

25. What is a VPS (Virtual Private Server)? [Електронний ресурс] - Режим доступу до ресурсу: https://aws.amazon.com/what-is/vps/?nc1=h_ls.

26. What is an SSL/TLS Certificate? [Електронний ресурс] - Режим доступу до ресурсу: https://aws.amazon.com/what-is/ssl-certificate/?nc1=h_ls.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ ОСНОВНИХ МОДУЛІВ

Telegram Bot

TelegramBot.java

```
package alex.telegram.bot.telegram;

import alex.telegram.bot.config.BotConfig;
import org.springframework.stereotype.Component;
import org.telegram.abilitybots.api.bot.AbilityWebhookBot;
import org.telegram.abilitybots.api.util.AbilityExtension;

import java.util.List;

@Component
public class TelegramBot extends AbilityWebhookBot {
    public TelegramBot(BotConfig botConfig,
List<AbilityExtension> extensions) {
        super(botConfig.getToken(), botConfig.getName(),
botConfig.getPath());
        addExtensions(extensions);
    }

    @Override
    public long creatorId() {
        return 1L;
    }
}
```

UserStartService.java

```
package alex.telegram.bot.service;

import alex.telegram.bot.message.model.UserUpdateEvent;
import
alex.telegram.bot.message.publisher.UserTelegramPublisher;
import alex.telegram.bot.model.User;
import alex.telegram.bot.repository.UserRepository;
```

```
import lombok.AllArgsConstructor;
import lombok.NonNull;
import org.springframework.stereotype.Service;

@Service
@AllArgsConstructor
public class UserStartService {
    private final UserRepository userRepository;
    private final UserTelegramPublisher userUpdatePublisher;

    public boolean exists(@NonNull Long futurityId) {
        return
userRepository.existsByFuturityUserId(futurityId);
    }

    public void processStart(@NonNull Long futurityId, @NonNull
Long chatId) {
        User user = new User(futurityId, chatId);
        userRepository.save(user);
        publish(futurityId);
    }

    private void publish(Long futurityId) {

userUpdatePublisher.publishEvent(UserUpdateEvent.of(futurityId))
;
    }
}
```

StartCommandHandler.java

```
package alex.telegram.bot.telegram.command;

import alex.telegram.bot.service.UserStartService;
import alex.telegram.bot.utils.BotUtils;
import alex.telegram.bot.validator.UserStartValidator;
import lombok.AllArgsConstructor;
```

```
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;
import org.telegram.abilitybots.api.objects.Ability;
import org.telegram.abilitybots.api.objects.Locality;
import org.telegram.abilitybots.api.objects.MessageContext;
import org.telegram.abilitybots.api.objects.Privacy;
import org.telegram.abilitybots.api.util.AbilityExtension;

@Slf4j
@Component
@AllArgsConstructor
public class StartCommandHandler implements AbilityExtension {
    private final UserStartValidator validator;
    private final UserStartService service;

    private static final String COMMAND = "start";
    private static final String SUCCESS_MESSAGE =
            "Thank you for using Futurity Bot, you will be
getting notification about deadline";

    public void handle(MessageContext context) {
        log.info("Got start event for user: {}",
context.user().getUserName());
        validator.validate(context);
        service.processStart(BotUtils.extractStartId(context),
context.chatId());
        log.info("User with futurity id={} and chatId={} has
been processed successfully",
                BotUtils.extractStartId(context),
context.chatId());

        BotUtils.sendMessage(SUCCESS_MESSAGE, context);
    }

    @SuppressWarnings("unused")
    public Ability startAbility() {
```

```
        return Ability.builder()
                .name(COMMAND)
                .info("Description")
                .locality(Locality.USER)
                .privacy(Privacy.PUBLIC)
                .action(this::handle)
                .build();
    }
}
```

UserNotificationSender.java

```
package alex.telegram.bot.service;

import alex.telegram.bot.client.TaskClient;
import alex.telegram.bot.message.model.Notification;
import alex.telegram.bot.model.Task;
import alex.telegram.bot.model.User;
import alex.telegram.bot.config.FuturityUrlConfig;
import alex.telegram.bot.repository.UserRepository;
import alex.telegram.bot.telegram.TelegramBot;
import lombok.AllArgsConstructor;
import lombok.NonNull;
import org.springframework.stereotype.Service;

@Service
@AllArgsConstructor
public class UserNotificationSender {
    private final UserRepository repository;
    private final TelegramBot bot;
    private final TaskClient taskClient;
    private final FuturityUrlConfig properties;

    private static final String MESSAGE =
            "Hello! There %s left to complete the task '%s'.
Hurry up! Here is a [link](%s) to the board";
```

```
    public void sendNotification(@NonNull Notification
notification) {
        User user = getUser(notification);
        bot.silent().sendMd(buildMessage(notification),
user.getChatId());
    }

    private String buildMessage(Notification notification) {
        Task info =
taskClient.getTaskInfo(notification.getTaskId(),
notification.getUserId());

        return MESSAGE.formatted(notification.getPart(),
info.getName(), properties.buildUrl(info.getProjectId()));
    }

    private User getUser(Notification notification) {
        return
repository.findByFuturityUserId(notification.getUserId())
                .orElseThrow(() ->
                        new
IllegalStateException(String.format("Failed to find user with
id=%s", notification.getUserId()))
                );
    }
}
```

Notification Service

ScheduleService.java

```
package com.alex.futurity.notificationservice.scheduler;

import
com.alex.futurity.notificationservice.scheduler.job.Notification
Job;
import
com.alex.futurity.notificationservice.scheduler.model.ScheduleRe
quest;
import com.alex.futurity.notificationservice.utils.DateContext;
import com.alex.futurity.notificationservice.utils.DateSplitter;
import com.alex.futurity.notificationservice.utils.DateUtils;
import
com.alex.futurity.notificationservice.utils.TaskConvertor;
import lombok.NonNull;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.quartz.*;
import org.springframework.data.util.Pair;
import
org.springframework.scheduling.quartz.SchedulerFactoryBean;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.List;

import static
com.alex.futurity.notificationservice.utils.JobExtractor.extract
Id;

@Slf4j
@Service
public class ScheduleService {
    private final Scheduler scheduler;

    public ScheduleService(SchedulerFactoryBean
schedulerFactory) {
```

```
        this.scheduler = schedulerFactory.getScheduler();
    }

    @SneakyThrows
    public void unscheduleTask(@NonNull Long taskId) {
        List<TriggerKey> keys = List.of(
                buildTriggerKey(taskId,
DateContext.DatePart.DAY),
                buildTriggerKey(taskId,
DateContext.DatePart.MINUTES)
        );

        log.info("Unscheduling task with id={}", taskId);
        scheduler.unscheduleJobs(keys);
        log.info("Task with id={} has been unscheduled",
taskId);
    }

    public void scheduleTask(@NonNull ScheduleRequest
scheduleRequest) {

DateSplitter.splitDate(scheduleRequest.getTimeToSchedule()).stre
am()
                .map(context -> Pair.of(
                        buildTrigger(context,
scheduleRequest.getTaskId()),
                        buildJobDetail(scheduleRequest,
context))
                )
                .forEach(pair -> scheduleJob(pair.getFirst(),
pair.getSecond()));
    }

    @SneakyThrows
    public void rescheduleTask(@NonNull ScheduleRequest
scheduleRequest) {
```

```
        unscheduleTask(scheduleRequest.getTaskId());
        scheduleTask(scheduleRequest);
    }

    @SneakyThrows
    private void scheduleJob(Trigger trigger, JobDetail
jobDetail) {
        log.info("Scheduling job with {} id for date {}",
extractId(jobDetail), trigger.getStartTime());
        scheduler.scheduleJob(jobDetail, trigger);
        log.info("Job with {} id has been scheduled",
extractId(jobDetail));
    }

    private static Trigger buildTrigger(DateContext context,
Long taskId) {
        Date startAt = DateUtils.toDate(context.getDeadline());

        return TriggerBuilder.newTrigger()

.withIdentity(TriggerKey.triggerKey(taskId.toString(),
context.getDatePart()))
                .startAt(startAt)
                .build();
    }

    private static JobDetail buildJobDetail(ScheduleRequest
scheduleRequest, DateContext context) {
        return JobBuilder.newJob(NotificationJob.class)
                .withDescription("Schedule notification")

.usingJobData(TaskConvertor.toJobDataMap(scheduleRequest,
context))
                .storeDurably()
                .build();
    }
```

```
    private static TriggerKey buildTriggerKey(Long taskId,
DateContext.DatePart datePart) {
        return TriggerKey.triggerKey(taskId.toString(),
datePart.getLeftTime());
    }
}
```

TaskConvertor.java

```
package com.alex.futurity.notificationservice.utils;

import com.alex.futurity.notificationservice.model.JobContext;
import com.alex.futurity.notificationservice.model.TaskInfo;
import lombok.NonNull;
import lombok.experimental.UtilityClass;
import org.quartz.JobDataMap;

import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Map;
import java.util.Objects;

@UtilityClass
public class TaskConvertor {
    private static final String TASK_KEY = "taskKey";
    private static final String USER_KEY = "userKey";
    private static final String DEADLINE_KEY = "deadlineKey";
    private static final String DEADLINE_PART_KEY =
"deadlinePartKey";

    private static final DateTimeFormatter FORMATTER =
DateTimeFormatter.ISO_OFFSET_DATE_TIME;

    public static <T extends TaskInfo> JobDataMap
toJobDataMap(@NonNull T info, DateContext context) {
        Map<String, String> map = Map.of(
```

```
                TASK_KEY, Long.toString(info.getTaskId()),
                USER_KEY, Long.toString(info.getUserId()),
                DEADLINE_KEY, formatDate(context.getDeadline()),
                DEADLINE_PART_KEY, context.getDatePart()
        );

        return new JobDataMap(map);
    }

    public static JobContext getContext(@NonNull JobDataMap map)
{
        String taskId =
Objects.requireNonNull(map.getString(TASK_KEY), "task id must be
present");
        String userId =
Objects.requireNonNull(map.getString(USER_KEY), "user id must be
present");
        String deadline =
Objects.requireNonNull(map.getString(DEADLINE_KEY), "deadline
must be present");
        String deadlinePart =
Objects.requireNonNull(map.getString(DEADLINE_PART_KEY),
"deadline must be present");

        return JobContext.builder()
                .taskId(Long.parseLong(taskId))
                .userId(Long.parseLong(userId))
                .timeToSchedule(parseDate(deadline))

.datePart(DateContext.DatePart.fromValue(deadlinePart))
                .build();
    }

    private static String formatDate(ZonedDateTime dateTime) {
        return FORMATTER.format(dateTime);
    }
```

```
    private static ZonedDateTime parseDate(String date) {
        return ZonedDateTime.parse(date, FORMATTER);
    }
}
```

TaskUpdateEventConsumer.java

```
package com.alex.futurity.notificationservice.message.consumer;

import
com.alex.futurity.notificationservice.message.model.UpdateTaskEv
ent;
import
com.alex.futurity.notificationservice.scheduler.ScheduleService;
import
com.alex.futurity.notificationservice.scheduler.model.ScheduleRe
quest;
import lombok.NonNull;
import lombok.Value;
import lombok.experimental.Delegate;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.function.Consumer;
import java.util.function.Predicate;

@Slf4j
@Component("taskUpdateConsumer")
public class TaskUpdateEventConsumer extends
TaskEventConsumer<UpdateTaskEvent> {
    private final List<ConditionCfg> conditions = List.of(
            ConditionCfg.of(UpdateTaskEvent::hasDeadline,
this::processDeadlineUpdate),
            ConditionCfg.of(UpdateTaskEvent::isDeadlineRemoved,
this::processDeadlineRemove),
```

```
            ConditionCfg.of(UpdateTaskEvent::getCompleted,
this::processTaskCompletion)
    );

    public TaskUpdateEventConsumer(ScheduleService
scheduleService) {
        super(scheduleService);
    }

    @Override
    protected void process(UpdateTaskEvent body) {
        conditions.stream()
                .filter(cfg -> cfg.test(body))
                .findFirst()
                .ifPresent(cfg -> cfg.accept(body));
    }

    private void processTaskCompletion(UpdateTaskEvent event) {
        log.info("Got completion task event for task={}",
event.getId());
        scheduleService.unscheduleTask(event.getId());
    }

    private void processDeadlineRemove(UpdateTaskEvent event) {
        log.info("Got remove deadline event for task={}",
event.getId());
        scheduleService.unscheduleTask(event.getId());
    }

    private void processDeadlineUpdate(UpdateTaskEvent event) {
        log.info("Got update deadline event for task={},
deadline='{}'", event.getId(), event.getDeadline());
        ScheduleRequest request = ScheduleRequest.builder()
                .userId(event.getUserId())
                .taskId(event.getId())
                .timeToSchedule(event.getDeadline())
```

```
                .build();

        scheduleService.rescheduleTask(request);
    }

    @Value(staticConstructor = "of")
    private static class ConditionCfg {
        @NonNull
        @Delegate
        Predicate<UpdateTaskEvent> predicate;

        @NonNull
        @Delegate
        Consumer<UpdateTaskEvent> consumer;
    }
}
```

DateSplitter.java

```
package com.alex.futurity.notificationservice.utils;

import lombok.NonNull;
import lombok.experimental.UtilityClass;

import java.time.ZonedDateTime;
import java.time.temporal.ChronoUnit;
import java.util.Collections;
import java.util.List;

@UtilityClass
public class DateSplitter {
    private static final long FIVE_MINUTES_DEADLINE = 5;
    private static final long ONE_DAY_DEADLINE = 1;

    public List<DateContext> splitDate(@NonNull ZonedDateTime
date) {
        if (isLaterThenOneDay(date)) {
```

```
            return List.of(oneDayContext(date),
fiveMinutesContext(date));
        }

        if (isLaterThenFiveMinutes(date)) {
            return List.of(fiveMinutesContext(date));
        }

        return Collections.emptyList();
    }

    private static DateContext fiveMinutesContext(ZonedDateTime
dateTime) {
        return
DateContext.of(dateTime.minusMinutes(FIVE_MINUTES_DEADLINE),
DateContext.DatePart.MINUTES);
    }

    private static DateContext oneDayContext(ZonedDateTime
dateTime) {
        return
DateContext.of(dateTime.minusDays(ONE_DAY_DEADLINE),
DateContext.DatePart.DAY);
    }

    private static boolean isLaterThenOneDay(ZonedDateTime
dateTime) {
        return ChronoUnit.DAYS.between(DateUtils.now(),
dateTime) >= 1;
    }

    private static boolean isLaterThenFiveMinutes(ZonedDateTime
dateTime) {
        return ChronoUnit.MINUTES.between(DateUtils.now(),
dateTime) >= 5;
    }
```

```
}
```

NotificationDeadlinePublisher.java

```
package com.alex.futurity.notificationservice.notification;

import lombok.AllArgsConstructor;
import lombok.NonNull;
import lombok.extern.slf4j.Slf4j;
import org.springframework.cloud.stream.function.StreamBridge;
import org.springframework.stereotype.Component;

@Slf4j
@Component
@AllArgsConstructor
public class NotificationDeadlinePublisher {
    private final StreamBridge streamBridge;

    private static final String BINDER_NAME =
"notificationPublisher";

    public void publish(@NonNull Notification notification) {
        log.info("Publishing notification for userId={} and
taskId={}", notification.getUserId(), notification.getUserId());
        streamBridge.send(BINDER_NAME, notification);
        log.info("Notification has been published for userId={}
and taskId={}", notification.getUserId(),
notification.getUserId());
    }
}
```

Project service

TaskEventPublisher.java

```
package com.alex.futurity.projectserver.message;

import com.alex.futurity.projectserver.context.UserContext;
import com.alex.futurity.projectserver.entity.Task;
```

```
import com.alex.futurity.projectserver.message.model.*;
import lombok.AllArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.cloud.stream.function.StreamBridge;
import org.springframework.messaging.Message;
import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Component;

@Slf4j
@Component
@AllArgsConstructor
public class TaskEventPublisher {
    private final StreamBridge streamBridge;
    private static final String BINDER_NAME = "taskPublisher";

    public void publishCreationEvent(Task task) {
        CreationTaskEvent event = CreationTaskEvent.builder()
                .id(task.getId())
                .deadline(task.getDeadline())
                .userId(UserContext.getUserId())
                .build();

        publish(event, TaskRoutingKey.CREATED);
    }

    public void publishUpdateEvent(Task task) {
        UpdateTaskEvent event = UpdateTaskEvent.builder()
                .id(task.getId())
                .deadline(task.getDeadline())
                .completed(task.isCompleted())
                .userId(UserContext.getUserId())
                .build();

        publish(event, TaskRoutingKey.UPDATED);
    }
```

```
    public void publishDeleteEvent(Task task) {
        DeleteTaskEvent event = DeleteTaskEvent.builder()
                .id(task.getId())
                .build();

        publish(event, TaskRoutingKey.DELETED);
    }

    private void publish(TaskEvent taskEvent, TaskRoutingKey
routingKey) {
        if (!UserContext.hasTelegram()) {
            return;
        }

        Message<TaskEvent> message = MessageBuilder
                .withPayload(taskEvent)
                .setHeader(TaskRoutingKey.HEADER_NAME,
routingKey.getRoutingKey())
                .build();

        streamBridge.send(BINDER_NAME, message);
        log.info("Task with id {} and key {} has been
published", taskEvent.getId(), routingKey.getRoutingKey());
    }
}
```

ContextInterceptor.java

```
package com.alex.futurity.projectserver.context;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;
import
org.springframework.web.context.request.RequestAttributes;
```

```
import
org.springframework.web.context.request.RequestContextHolder;
import
org.springframework.web.context.request.ServletRequestAttributes
;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
public class ContextInterceptor extends OncePerRequestFilter {
    private static final String HAS_TELEGRAM_HEADER_NAME =
"telegram";
    private static final String USER_ID_HEADER_NAME = "user_id";

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
        boolean hasTelegram =
Boolean.parseBoolean(request.getHeader(HAS_TELEGRAM_HEADER_NAME)
);
        Long userId =
Long.valueOf(request.getHeader(USER_ID_HEADER_NAME));
        User user = User.builder()
                .hasTelegram(hasTelegram)
                .userId(userId)
                .build();
        ServletRequestAttributes requestAttributes = new
ServletRequestAttributes(request);
        requestAttributes.setAttribute(UserContext.USER, user,
RequestAttributes.SCOPE_REQUEST);


RequestContextHolder.setRequestAttributes(requestAttributes);
```

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

ColumnDao.java

```
package com.alex.futurity.projectserver.dao;

import com.alex.futurity.projectserver.dto.CreationTaskDto;
import com.alex.futurity.projectserver.dto.ProjectColumnDto;
import com.alex.futurity.projectserver.entity.ProjectColumn;
import com.alex.futurity.projectserver.entity.Task;
import
com.alex.futurity.projectserver.exception.ClientSideException;
import com.alex.futurity.projectserver.model.UserProject;
import com.alex.futurity.projectserver.repo.ColumnRepository;
import jakarta.transaction.Transactional;
import lombok.AllArgsConstructor;
import lombok.NonNull;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Repository;

import java.util.Collection;
import java.util.List;
import java.util.Objects;
import java.util.Optional;
import java.util.stream.Stream;

@Repository
@AllArgsConstructor
public class ColumnDao {
    private final ColumnRepository columnRepo;
    private static final String NOT_FOUND_MESSAGE = "The column
is associated with such data does not exist: %s";
    public List<ProjectColumnDto> getColumns(@NonNull
UserProject project) {
        return getColumnsByProject(project)
                .stream()
```

```
                .map(ProjectColumnDto::fromProjectColumn)
                .toList();
    }
    public List<ProjectColumn> getColumnsByProject(UserProject
project) {
        return columnRepo.findAllByProject(project.getUserId(),
project.getProjectId());
    }

    @Transactional
    public Optional<ProjectColumn> deleteColumn(@NonNull
UserProject project, @NonNull Long columnId) {
        List<ProjectColumn> columns =
getColumnsByProject(project);

        return columns.stream()
                .filter(column -> Objects.equals(column.getId(),
columnId))
                .findFirst()
                .map(column -> deleteAndShift(column, columns));
    }
    @Transactional
    public Task addTaskToColumn(@NonNull CreationTaskDto
creationTaskDto, @NonNull Long columnId) {
        return columnRepo.findById(columnId)
                .map(column -> column.addTask(new
Task(creationTaskDto.getName(), creationTaskDto.getDeadline(),
column)))
                .orElseThrow(() -> buildException(columnId));
    }
    @Transactional
    public void changeColumnName(@NonNull Long columnId, String
columnName) {
        columnRepo.findById(columnId)
                .ifPresent(column ->
column.setName(columnName));
```

```
    }
    @Transactional
    public List<Task> markColumnAsDone(@NonNull Long
columnToMark, Long columnToUnmark) {
        Stream<Task> completedTasks =
columnRepo.findById(columnToMark)
                .map(column -> column.setDoneColumn(true))
                .map(ProjectColumn::getTasks)
                .stream()
                .flatMap(Collection::stream)
                .map(task -> task.setCompleted(true));
        Stream<Task> uncompletedTasks =
Optional.ofNullable(columnToUnmark)
                .flatMap(columnRepo::findById)
                .map(column -> column.setDoneColumn(false))
                .map(ProjectColumn::getTasks)
                .stream()
                .flatMap(Collection::stream)
                .map(task -> task.setCompleted(false));
        return Stream.concat(completedTasks, uncompletedTasks)
                .toList();
    }
    private ProjectColumn deleteAndShift(ProjectColumn
columnToDelete, List<ProjectColumn> columns) {
        columnRepo.delete(columnToDelete);
        columns.stream()
                .filter(column -> column.getIndex() >
columnToDelete.getIndex())
                .forEach(column ->
column.setIndex(column.getIndex() - 1));

        return columnToDelete;
    }
    private static ClientSideException buildException(Long
columnId) {
```

```
        return new
ClientSideException(NOT_FOUND_MESSAGE.formatted(columnId),
HttpStatus.NOT_FOUND);
    }
}
```

Futurity UI

telegram-form.component.ts

```
import {AfterViewInit, Component, Inject} from '@angular/core';
import {NgbActiveModal} from "@ng-bootstrap/ng-bootstrap";
import {DOCUMENT} from "@angular/common";
import {UserInfoDto} from "../shared/dto/user-info-dto";

@Component({
  selector: 'app-telegram-form',
  templateUrl: './telegram-form.component.html',
  styleUrls: ['./telegram-form.component.css']
})
export class TelegramFormComponent implements AfterViewInit {
  userInfo: UserInfoDto;
  constructor(public activeModal: NgbActiveModal,
@Inject(DOCUMENT) private document: Document) { }
  ngAfterViewInit() {
    const dialog = this.document.querySelector(".modal-dialog")
as any;
    const content = this.document.querySelector(".modal-
content") as any;

    dialog.style.width = "350px";
    dialog.style.margin = "auto";
    content.style.borderRadius = "25px";
    content.style.background = "#282c35";
  }
  buildUrl() {
    return "https://t.me/FuturityTestBot?start=" +
this.userInfo.id;
  }
```

```
}
```

telegram-form.component.html

```
<div class="modal-header text-center">
  <button type="button" class="close border-0"
(click)="activeModal.close()">
    <span aria-hidden="true">&times;</span>
  </button>
</div>
<div class="w-100 d-flex flex-column align-items-center justify-
content-center background">
  <img class="img-fluid img-width" src="assets/telegram.png">
  <a class="btn mt-4" *ngIf="!userInfo.hasTelegram"
[href]="buildUrl()">Join Telegram</a>
  <button class="btn mt-4 disabled cursor-default"
*ngIf="userInfo.hasTelegram">You have joined to
Futurity</button>
</div>
<div class="modal-footer mt-2"></div>
```

telegram-form.component.css

```
:host ::ng-deep .modal-dialog {
  margin: auto !important;
}

.close {
  outline: none;
  color: #FFFF00;
  font-size: 28px;
}

.modal-header {
  border-top-left-radius: 20px;
  border-top-right-radius: 20px;
  background: #282c35;
  border: none;
```

```
  height: 20px;
}

.btn {
  margin-right: 0;
}

.background {
  background: #282c35;
}

.img-width {
  width: 200px;
}

.cursor-default {
  cursor: default;
}

.modal-footer {
  border: none;
  border-bottom-left-radius: 20px;
  border-bottom-right-radius: 20px;
}
```

Jenkins

javaPipeline.groovy

```
def call() {
    pipeline {
        agent any

        options {
            skipDefaultCheckout()
        }
        stages {
            stage('Stop running builds for the branch') {
```

```
        steps {
            stopRunningBuilds()
        }
    }

    stage('Wipe workspace and checkout') {
        steps {
            checkoutCurrentBranch()
        }
    }

    stage('Package and Push') {
        stages {
            stage('Publish build check') {
                steps {
                    publishChecks(
                            name: 'Build image',
                            title: 'Building image',
                            status: 'IN_PROGRESS'
                    )
                }
            }

            stage('Package') {
                steps {
                    runWithMaven {
                        sh 'mvn clean package'
                    }
                }
            }

            stage('Build docker image and push') {
                steps {
                    buildAndPublish()
                }
            }
```

```
                    stage('Trigger deploy job') {
                        steps {
                            triggerDeploy()
                        }
                    }
                }

                post {
                    always {
                        script {
                            def isBuildFailure =
currentBuild.result == 'FAILURE'
                            def stageStatus = isBuildFailure ?
'FAILURE' : 'SUCCESS'
                            def message = isBuildFailure ?
                                    'Failed to build your image'
:
                                    "Your image has been build
and published: ${dockerUtils.getImageNameAndTag()}"

                            publishChecks(
                                    name: 'Build image',
                                    title: 'Building image',
                                    summary: message,
                                    status: 'COMPLETED',
                                    conclusion: stageStatus
                            )
                        }
                    }
                }
            }
        }
    }
}
```

JenkinsfileDeploy.groovy

```
@Library('futurityPipelineLib') _
pipeline {
    agent none

    parameters {
        string(name: 'imageTag', trim: true, description: 'Image
tag to be deployed')
        string(name: 'imageName', trim: true, description:
'Image name to be deployed')
    }

    options {
        disableConcurrentBuilds()
    }

    stages {
        stage('Validate') {
            steps {
                script {
                    required(params.imageName, 'Image name is
not present')
                    required(params.imageTag, 'Image tag is not
present')
                }
            }
        }
        stage('Approve Deploy') {
            steps {
                timeout(time: 5, unit: 'MINUTES') {
                    input(message: "Deploy
'${params.imageName}:${params.imageTag}'?", ok: 'Deploy')
                }
            }
        }
```

```
        stage('Deploying') {
            agent any

            stages {
                stage('Preparing workspace') {
                    steps {
                        script {
                            withSsh {
                                sshUtils.prepareWorkspace(it)
                            }
                        }
                    }
                }
                stage('Up service') {
                    steps {
                        script {
                            withSsh {
                                sshUtils.upService(it,
params.imageName, params.imageTag)
                            }
                        }
                    }
                }
            }
            post {
                always {
                    script {
                        withSsh {
                            echo 'Cleaning up'
                            sshUtils.clearWorkspace(it)
                        }
                    }
                }
            }
        }
    }
```

```
}
def required(def param, def errorMessage) {
    if (!param) {
        error(errorMessage)
    }
}
```