ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Науково-навчальний інститут комп'ютерних наук і технологій
Кафедра прикладної математики і інформатики

«До захисту допущено»
Завідувач кафедри ПМІ
О.А. Дмитрієва
(підпис) (ініціали, прізвище)
«_____» _____ 2020 р.

Випускна кваліфікаційна робота

магістра

(освітньо-кваліфікаційний рівень)

на тему
«БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В РОЗПОДІЛЕНІЙ СИСТЕМІ ГРУПОВОЇ ВЗАЄМОПІДТРИМКИ РЕАЛІЗАЦІЇ ОСОБИСТИХ ЗАВДАНЬ»

Виконав: студент _2_ курсу, групи ___ІПЗм-19_____
(шифр групи)
напряму підготовки (спеціальності) 121 Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Омельченко І. А. _____
(прізвище та ініціали) (підпис)

Керівник _____д.т.н., проф. Дмитрієва О. А_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Покровськ – 2020

ДВНЗ «Донецький національний технічний університет»
Факультет Науково-навчальний інститут комп'ютерних наук і технологій
Кафедра прикладної математики та інформатики
Освітній ступінь магістр
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____О.А. Дмитрієва
(підпис) (ініціали, прізвище)
«_____» _____ 2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Омельченку Івану Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Балансування навантаження в розподіленій системі групової взаємопідтримки реалізації особистих завдань»
Спецчастина
Керівник роботи Дмитрієва О. А., д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом від « 28» вересня 2020 року № 727
2. Строк подання студентом роботи
3. Вихідні дані до роботи результати переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1) Визначення основних понять щодо проблематики у постановці особистих завдань та до
2) Аналіз існуючого програмного інструментарію для допомоги у виконанні особистих завдань, вивід критеріїв для їх порівняння та створення на основі отриманих критеріїв концепту розроблюваного сервісу
3) Аналіз та обрання програмного інструментарії для розробки сервісу
4) Розробка детального концепту сервісу
5) Розробка архітектури та конструювання основного функціоналу сервісу
6) Розробка архітектури для розподілення навантаження на окремі компоненти розроблюваного сервісу та її запровадження
7) Проведення навантажувального тестування отриманого сервісу
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
робота містить _____ таблиць, _____ рисунків, та іншого графічного матеріалу у кількості, достатній для розкриття сутності і демонстрації основних результатів кваліфікаційної роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літератури з предметної області	3.09.2020 – 27.09.2020	
2	Проведення аналізу проблеми виконання особистих задач та супутнього інструментарію, а також інструментарію для розробки веб-сервісу	28.09.2020 – 05.10.2020	
3	Розробка детального концепту та загальної архітектури та схеми БД сервісу	06.10.2020 – 18.10.2020	
4	Конструювання основного функціонала сервісу	19.10.2020 – 10.11.2020	
5	Запровадження механізмів горизонтального масштабування та завантаження отриманої конфігурації Kubernetes до хмарного обчислювального сервісу	11.10.2020 – 15.11.2020	
6	Оформлення пояснювальної записки	15.11.2020 – 30.11.2020	
7	Нормоконтроль пояснювальної записки, її перевірка антиплагіатною системою	01.12.2020 – 12.12.2020	
8	Оформлення супутніх матеріалів (розробка графічного матеріалу, написання доповіді, тощо) та рецензування роботи	13.12.2020 – 21.12.2020	
9	Захист роботи	22.12.2020	

Студент _____ Омельченко І. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Дмитрієва О. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Омельченко І. А. Балансування навантаження в розподіленій системі групової взаємопідтримки для реалізації особистих завдань / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення». – ДВНЗ ДонНТУ, Покровськ, 2020.

Об’єкт дослідження – процеси балансування навантаження в розподіленій системі організації особистих поточних справ користувачів.

Предметом дослідження є програмні інструменти підтримки мотивації при досягненні окремих цілей користувачів з використанням групової взаємопідтримки.

Метою роботи є розробка розподіленої системі сервісу групової взаємопідтримки виконання особистих завдань у форматі веб-застосунку з використанням технологій горизонтального масштабування навантаження зі сторони сервера, та розробки клієнтської частини у форматі SPA.

В процесі роботи проаналізовано сучасні програмні та інструментальні платформи, визначено підходи до візуалізації прогресу в завданні сервісу групової взаємопідтримки, та фактори, які впливають на досягнення цілей. Проведено порівняльний аналіз існуючих сервісів допомоги. Спроектовано архітектуру розроблюваного сервісу, визначено структуру особистого профілю користувача. Розроблено систему автоматичного підбору користувачів, календар завдань, систему обміну повідомленнями. Програмним результатом роботи є веб-застосунок, що складається з клієнтської частини у вигляді SPA і розподілене серверне забезпечення з функціоналом балансування навантаження (горизонтальне масштабування). Практичне значення роботи полягає у можливості використання отриманого веб-застосунку для рішення особистих завдань користувачів, що мають проблеми у аспектах мотивації, встановлення цілей та планування.

Ключові слова: мотивація, accountability, веб сервіс, масштабування, розподілена система, хмарні обчислення, Kubernetes, AJAX, SPA, Vue, fastAPI, Websocket, asyncio, NoSQL.

ABSTRACT

Omelchenko I. Load balancing in a distributed system of group mutual support for implementation of personal tasks / Graduation qualification work for obtaining an educational degree “Master” in speciality 121 “Software engineering”. – DonNTU, Pokrovsk, 2020.

The research object is the processes of load balancing in a distributed system of organization of users’ personal tasks.

The study’s subject is software tools for supporting motivation in achieving personal goals of users using group mutual support.

The purpose of the work is to develop a distributed system of the group support for achieving personal tasks in the format of a web application using horizontal scaling technologies at the server-side and an approach to developing the client part in the SPA format.

In the process of work were analyzed modern software and tool platforms, were determined approaches to the visualization of tasks progress within a group of mutual support, and factors that influence the achievement of goals. Was conducted a comparative analysis of existing assistance services. Were designed architecture of the developed service and defined a structure of the personal profile of the user. Also, a system of automatic selection of users, a calendar of tasks, and a system of exchange of messages were developed.

The resulting software is a web application consisting of a client part as SPA and distributed server software with load balancing functionality (horizontal scaling). The practical significance of the work is the possibility of using the resulting web application to solve personal tasks of users who have problems in terms of motivation, goal setting, and planning.

Keywords: motivation, accountability, web service, scaling out, distributed system, cloud computing, Kubernetes, AJAX, SPA, Vuejs, fastAPI, Websocket, asyncio, NoSQL

ЗМІСТ

Перелік умовних позначень, символів, скорочень і термінів.....8
Вступ.....9
Розділ 1 Аналіз та постановка задачі.....11
1.1 Підходи та інструменти для постановки та контролю досягнення цілей 12
1.1.1 Візуалізація прогресу.....13
1.1.2 Фактор негативної мотивації.....13
1.1.3 Фактор публічної звітності.....14
1.1.4 Порівняння існуючих сервісів допомоги у виконанні задач.....15
1.2 Порівняння існуючих сервісів. Виведення концепту більш досконалого сервісу.....16
1.3 Інструменти та програмні платформи для розробки веб-сервісу.....19
1.3.1 Програмні платформи створення клієнтських веб-інтерфейсів.....20
1.3.2 Протокол передачі повідомлень для чата.....21
1.3.3 Асинхронні веб-сервера.....23
1.3.4 Вибір між БД типу SQL та NoSQL.....27
1.3.5 Контейнеризація засобами Docker.....29
1.4 Керування контейнерами засобами Kubernetes.....30
1.5 Висновки за розділом.....32
Розділ 2 Розгорнутий концепт проекту розроблюваного сервісу.....34
2.1 Основні логічно-функціональні компоненти.....34
2.1.1 Реєстрація та авторизація.....35
2.1.2 Особистий профіль користувача.....36
2.1.3 Особистий перелік розділів, проектів та задач.....38
2.1.4 Пошук проектів інших користувачів для кооперації.....42
2.1.5 Чат.....45
2.1.6 Система повідомлень.....48

2.1.7 Система автоматичного підбору користувачів.....50
2.1.8 Календар задач.....50
2.2 Висновки за розділом.....51
Розділ 3 Архітектура і особливості реалізації сервісу.....53
3.1 Загальна архітектура сервісу.....53
3.1.1 Серверна частина.....54
3.1.2 Клієнтська частина.....54
3.2 Схема даних БД.....55
3.2.1 Особливості схеми колекції користувачів.....56
3.2.2 Особливості колекції чатів.....58
3.3 Схема точок доступу API сервера.....59
3.3.1 Схема HTTP запитів.....59
3.3.2 Схема Websocket запитів.....65
3.4 Компоненти клієнтського інтерфейсу.....67
3.4.1 Маршрутизація компонентів.....67
3.4.2 Використання менеджеру станів даних Vuex.....68
3.5 Висновки за розділом.....70
Розділ 4 Масштабування та розгортання серверної частини.....71
4.1 Розбиття на мікросервіси.....71
4.1.1 Масштабування чату з використанням патерна Publisher-Subscriber 72
4.2 Схема розміщення мікросервісів у Kubernetes.....73
Висновки.....75
Додаток А Зауваження нормконтролера.....81
Додаток Б Порівняння сервісів.....83
Додаток В Вихідні коди застосунку.....88
Додаток Г Презентаційний матеріал.....95

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД	База даних
РСКБД	Реляційна система керування баз даних
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
SPA	Single Page Applicatio

ВСТУП

Кваліфікаційна робота містить: 104 сторінки, 21 рисунок, 8 таблиць, 4 додатка, 51 використане джерело.

Дана робота розглядає проблему виконання особистих задач. Ця тема постає особливо гостро, якщо розглядати її у контексті сучасного перевантаженого інформацією світі, світі постійних змін, легких задоволень та соціальних мереж, що зображають недосяжні ідеали, тим самим змушуючи користувачів ставити відповідні недосяжні стандарти, а відповідно і цілі. Таким чином отримуємо величезну кількість людей що не можуть ефективно організувати щоденну рутину.

Ринок також йде за трендом, тож маємо значну кількість матеріалу тематики так званого тайм менеджменту (методи керування справами та часом), а також набір інструментарію для організації особистих поточних задач.

Самі по собі інструменти з організації особистих справ не є чимось новим, але їх поєднання з використанням аспекту соціалізації є досить відкритою темою для подальшого вдосконалення, на що й націлено дану роботу.

У роботі проведено аналіз існуючих програмних рішень, як суто для організації справ, так і ті, що в тій чи іншій мірі залучують соціальний аспект. На основі цього аналізу було запропоновано концепт сервісу, що поєднує найкращі аспекти з досліджених сервісів.

Вочевидь, що найкращим форматом для такої системи є веб-застосунок що потребує встановлення та потенційно доступний на будь-якому пристрої, проте у даній роботі зроблено акцент саме на PC платформі задля спрощення розробки.

Розглянуто використанням сучасного підходу розробки клієнтських веб-інтерфейсів SPA. Значна частина роботи присвячена побудові розподіленої архітектури сервера з використанням публічних хмарних сервісів, таким чином потенційно забезпечуючи одночасну обробку великої кількості користувачів.

Новизна розроблюваного сервісу заключена у збільшенні фокуса на використання веб-чату як головного елементу кооперації, а також на фокусі таких кооперацій на конкретних проектах задач. Такі кооперації існують лише на час виконання проектів, таким чином підтримуючи максимальний фокус на процесі виконання конкретного проекту, не змішуючи декілька тем одночасно.

В першому розділі наведено аналіз існуючих сервісів мобільної та веб-платформи, виведено основні вимоги до розроблюваного сервісу, а також огляд та вибір доцільних програмних засобів його реалізації.

У другому розділі вимоги до сервісу з першого розділу розвинено в досить детальний концепт з розподілом сервісу на перелік логічно-функціональних компонентів.

У третьому розділі розглянуто основні аспекти архітектури розроблюваного сервісу з фокусом на головні функціональні вимоги, минуючи проблеми масштабування. Наведено схему БД та опис API для HTTP та Websocket (у вигляді подій) протоколу.

РОЗДІЛ 1
АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

У сучасному динамічному світі безмежних можливостей і не менш безмежних відволікаючих чинників питання постановки правильних цілей, ефективного режиму, контролю часу, організації справ і т.п. стає все більш актуальним. Наприклад, люди, яким зараз 20-24 роки в 3 рази більше схильні до зміни місця роботи, ніж ті, яким зараз 45-54 [24]. Причому, судячи з постійного зростання ринку індустрії самодопомоги, різних технік управління часу, коучінгу (або менторства, як особистого, так і корпоративного), додатків з обліку часу і організації справ (що досягає вартості понад 10 мільярдів доларів) [33], для більшості задовільних і робочих відповідей так і не знаходиться.

Можна виділити деякі проблеми [30] вищеописаних незадоволених споживачів:

– надані методи занадто складні в застосуванні, їх складно вписати в реальне життя;

– бажання отримати результат негайно, отже швидке розчарування;

– постійне споживання нових інформаційних матеріалів з самопічі, замість прийняття реальних дій;

– неякісне застосування наданих методів з подальшим розчаруванням у них;

– нестача або відсутність зовнішньої мотивації, соціальної підтримки.

Втім, у даній роботі розглянуто специфічну, але, імовірно, одну з найважливіших тем – постановка і досягнення особливих цілей з великим наголосом на саме процес їх досягнення.

1.1 Підходи та інструменти для постановки та контролю досягнення цілей

Слід розрізняти поняття цілі та завдання. Ціль орієнтована на кінцевий результат (відповідає на питання «що?»), може бути кілька абстрактної, тоді як завдання орієнтована на процес, виконання конкретних кроків (відповідає на питання «як?»).

Під цілями можуть матися на увазі:

- виконання деякого разового завдання;
- виконання деякого проекту, тобто комплексу завдань;
- закріплення звички (або відмови від негативної).

Найчастіше, завдання, шляхом декомпозиції, формуються з цілей або більш абстрактних понять, на зразок прагнень, бажань, мрій, ідей. Однак, в кінцевому підсумку потрібно мати конкретне завдання – комплекс елементів з прикладених зусиль і ресурсів (матеріальних, людських, знань), часу і термінів, очікуваного результату, мотивації, а також контексту завдання (головної мети, однорівневих, залежних і залежних завданнях). Причому, іноді отримані завдання вимагають подальшої декомпозиції – питання скоріше індивідуальне та творче.

Невірно визначені цілі і завдання – одна з головних запорук невдачі. Для вирішення цієї проблеми існують методи так званого цілепокладання [51]. Одним з популярних його методів є метод S.M.A.R.T [35].

Ключовою особливістю даної роботи буде фокус на мотиваційному аспект,і а саме поняття звітності – звітності як суто перед собою, так і перед іншими людьми, що може значно посилити ефект: наявність фактора звітності перед іншими людьми приблизно на третину підвищує шанс на успішне досягнення поставленої мети [32].

Далі будуть розглянуті деякі підходи для підвищення даного мотиваційного аспекту та приклади сервісів, що їх активно використовують.

1.1.1 Візуалізація прогресу

Суть даного підходу (аспекту) в наданні більш відчутного візуального представлення прогресу. Надає як додаткове підтвердження завершення завдання (сам момент викреслювання або відмічання завдання як виконаного у переліку завдань), так і огляд вже виконаних завдань в сукупності. Підсилює як відчуття від досягнення (зокрема відчуття прогресу), так і мотивацію продовжувати. Прикладами можуть бути:

- простий чек-лист як сам по собі, так і виведений зі структури завдань перелік на день (що реалізує todoist.com);
- статистика дотримування поставленого графіка – кількість «перемог» поспіль, статистика успішних днів в місяць (особливо для щоденних звичок);
- графік змін деякого параметра за днями (наприклад, при боротьбі із зайвою вагою);
- рейтинг (наприклад, в питанні вивчення іноземних мов, наприкулад як в сервісі clozemasters.com).

До цієї ж категорії можна віднести аспект гейміфікації на прикладі сервісу habitica.com [34].

1.1.2 Фактор негативної мотивації

Можна штучно додати негативну мотивацію. Розповсюдженим підходом є публічне заявлення про своє прагнення (окремий випадок цього підходу буде розглянуто в підрозділі про звітність). Однак, часто куди більш дієвим способом є закладання чогось у заставу, наприклад деяку грошову суму товаришеві. Існують сервіси, що автоматизують (заразом комбінуючи з вищеописаними методами з підрозділу про візуалізації прогресу) даний підхід, зокрема stickk.com та beeminder.com. Користувач вказує цілі, умови

виконання та суму застави, яка автоматично списується з рахунку в разі порушення умов виконання мети, заодно надаючи графік виконання.

Додатково є фактор соціалізації, прикладом якого є відчуття при спостеріганні процесу досягнення цілей іншими користувачами системи, або при публікації власного.

1.1.3 Фактор публічної звітності

Дієвість даного фактора заснована на соціальній відповідальності – поняттях по типу «відповідальність за слово», персональному іміджі і тому подібне. Звичним підходом є повідомити нікому людині (другу чи знайомому, якоїсь публіці) про свої наміри, що створює негативну мотивацію аналогічну описаної у пункті 1.1.2. На жаль, досить часто це стосується якоїсь разової мети, а обговорення далі самого факту постановки мети не йдуть.

Однак цей найпримітивніший варіант підходить далеко не всім. Більш дієвим способом є довготривалий регулярний вид звітності по конкретним атомарним задачам у контексті деякої значної цілі. Даний вид взаємодії можна умовно розділити на наступні види:

1) одностороння відмова звітність (наприклад, сервіси accountably.net та bossasaservice.life);

2) взаємна звітність (наприклад, habitshareapp.com).

Перший вид орієнтується на процес, на виконання поставлених завдань. Другий орієнтується на прогрес, може завдавати куди більший вплив; підключає змагальний і глибший аналітичний аспект (звісно, все залежить від конкретного випадку) [1].

Втім, для другого виду не завжди просто підібрати відповідного компаньйона (або групу), як по інтересам і вмінням, так і за якістю прогресу за бажаними цілями. Засобом пошуку можуть слугувати соціальні мережі, наприклад тема на Reddit https://www.reddit.com/r/GetMotivatedBuddies.

Схожими рисами володіє підхід роботи з особистим тренером або наставником (часто вживаються англіцизми як коуч та ментор). Цей підхід може мати куди більшу ефективністю ніж вищеописані, але знайти відповідну людину для роботи ще складніше, а головне – дорожче. Прикладом такого онлайн-сервісу може слугувати coach.me.

Окремо варто виділити сервіси по типу smartprogress.do. Даний сервіс надає структурну розбивку цільового проекту на цілі і завдання із зазначенням термінів, що візуалізується в діаграмі Ґанта [50]. Більше того, передбачено щоденник з можливістю коментування, коментування будь то конкретними запрошеними користувачами чи відкритою публікою. Є сторінка з публічно доступними цілями інших користувачів. Виконання цілей можна підкріпити платною заставою. Є окремі сторінки для пошуку наставника (виконання конкретної мети в рамках групи), а також загального переліку користувачів з рейтингом, заснованим на кількості виконаних користувачем цілей.

1.1.4 Порівняння існуючих сервісів допомоги у виконанні задач

Далі наведено порівняння конкретних аспектів, які реалізують раніше описані сервіси – позитивні і негативі. Використовуючи отримані результати, стає можливим уявити дещо кращу комбінацію аспектів для розроблюваного сервісу. Отримана порівняльна таблиця наведена у додатку Б.

Порівняння проводилося за такими ознаками:

– «структуризація» – спосіб структурування цілей: разові, або що повторюються (звички), можливість додавання підзадачі і т.п.;

– «візуалізація прогресу» – наявність різноманітних графіків, статистик і звітів;

– «грошова застава» – можливість налаштування грошового покарання за порушення графіка виконання цілей;

– «групова робота і соціальна відповідальність » – наявність різних засобів для демонстрації та обговорення прогресу (персонально або у групі);

– «пошук компаньйонів» – можливість пошуку компаньйонів, наставників або перевіряючих звіти про досягнення цілей;

– «простота у використанні» – наскільки зручний інтерфейс користувача;

– «вартість» – вартість разових платних послуг, або построкових підписок.

Важливо зауважити, що порівняння проводилося з точки зору усередненого користувача яким його собі уявляє автор. Крім того, для різниих категорій користувачів необхідний і достатнім вважається різний функціонал. Проте, теоретично, більш досконалий сервіс зможе забезпечити підтримку ширшої аудиторії шляхом абстракції або надання більш тонких налаштувань за потребою, дозволяючи налаштовуватися на конкретного користувача.

Виходячи з наданого набору критеріїв, можна відзначити сервіс smartprogress.do як має максимально багатий функціонал. З іншого боку, можна виділити сервіс accountably.net надає нічого більш ніж чат (втім, в планах розробника є як мінімум додавання простого todo переліку завдань), таким чином встановлюючи фокус конкретно на спілкуванні, але не нав’язуючи свого інструментарію з організації задач.

1.2 Порівняння існуючих сервісів. Виведення концепту більш досконалого сервісу

У процесі аналізу отриманої порівняльної таблиці з додатку Б, було виведено наступний перелік функціональних вимог до вдосконаленого сервісу:

а) структуризація завдань:

1) розбиття на завдання верхнього рівня – цілі, котрими можуть бути як разові завдання, так і звички, причому разові завдання, як і звички, можуть мати відсоток виконання, а звички можуть позначатися входженням в діапазон цільових значень (в тому числі і негативний – для звичок типу утримання);

2) наявність функціоналу для завдання повторень завдань (особливо для звичок), причому, у разі невиконання завдання-звички, завдання може бути або перенесене на наступний день (накопичення боргу), або просто бути відзначеним як провалене;

3) звички можуть мати умову завершення, тобто коли контроль більше не потрібен – за досягненням деякої кількості виконання (або невиконання), за досягненням деякої іншої цілі;

4) цілі верхнього рівня можуть включати в себе звички як вкладені цілі;

5) наявність бази заздалегідь заготовлених планів з розбивкою за категоріями;

6) вказівка параметрів пріоритетності і складності;

7) можливість перемикання формату заповнення завдань для застосування методик постановки задач, наприклад S.M.A.R.T;

8) наявність щоденника для цілей вищого порядку;

б) візуалізація прогресу:

1) для цілей – діаграма Ганта;

2) наявність календаря, де кожен день зображений у вигляді зафарбованою двійковій секторної діаграми відповідної відсотку виконання завдань за день (або у іншому вигляді) – як для окремих завдань, так і для сукупності всіх;

3) аналогічно описаному вище підпункту переліку, відображення кількісного прогресу у вигляді лінійного графіка;

4) підрахунок статистик стійкості виконання звичок;

5) елемент гейміфікації – рейтинг, що ґрунтується виключно на дотриманні графіка виконання завдань, у разі ж обґрунтування рейтингу показником абсолютної кількості виконаних завдань, може виникнути бажання додати більше завдань, що може призвести до деяких негативних наслідків, на зразок поверхневого виконання завдань і емоційного вигорання;

6) ведення статистики успішного виконання завдань К-днів без невдач поспіль, а також аналогічної статистика для умови «без двох невдалих днів поспіль»;

в) грошова застава;

1) можливість призначати цілям плату за невиконання, причому плата проводиться відразу і повертається в разі успіху;

2) можливість запрошення спостерігача.

г) групова робота і соціальна відповідальність:

1) наявність профілів, сторінки друзів і чата (в тому числі групового);

2) звіт на сторінці профілю переліку цілей (з налаштуваннями приватності), активності користувача в хронологічному порядку,

3) наявність сторінки агрегованої активності друзів;

4) за замовчуванням вищевказана сторінка порожня, користувач повинен сам додавати цікавлячі активності його друзів – сервіс повинен бути інструментом, який користувач використовує за своїм конкретним наміром не відволікаючись на непотрібну йому інформацію;

5) користувач може залишити питання на сторінці власної мети, що буде відображено в окремій стрічці питань всіх користувачів (або тільки із переліку друзів);

6) можливість спільного редагування цілей окремого користувача.

д) пошук компаньйонів:

1) пошук користувачів за назвою цілей;

2) пошук по імені, країні, віковою діапазону, статі;

3) користувач може вказати зацікавленість або її відсутність щодо пошуку компаньйонів як в цілому за профілем, так і за окремими цілями;

е) простота у використанні:

1) акцент інтерфейсу на наступній атомарній задачі, відображення мінімуму необхідних елементів на головній формі, все інше – по посиланнях на інші сторінки.

ж) вартість:

1) увесь автоматичний функціонал за символічну сплату;

2) ментори самі собі встановлюють вартість.

Даний перелік вимог представляє мало нового в порівнянні з тим же Smartprogress.do, але ключова різниця є у фокусі на спілкування протягом усього процесу досягнення мети. Робота з компаньйонам максимально нав’язлива. Пошук і додавання компаньйона (спостерігача або наставника, в межах декількох осіб) є першим кроком разом зі створенням спеціальної тематичної кімнати в чаті, після чого йтиме обговорення і постановка цілей для кожного учасника. Таким чином, у процесі спільного обговорення можна знизити ймовірність постановки невірних і надлишкових цілей, краще зрозуміти мотивацію і прагнення учасників, їх проблеми.

1.3 Інструменти та програмні платформи для розробки веб-сервісу

При створенні сучасного веб-сервісу можна виділити наступні ключові компоненти:

- клієнтський інтерфейс;
- серверний додаток обробки запитів;
- база даних.

Також, створення хоча б декілька масштабового проекту важко уявити без інструментів CI/CD, таких як:

- система контролю версій;
- інструменти контейнеризації.

1.3.1 Програмні платформи створення клієнтських веб-інтерфейсів

Перш за все слід зазначити, що сучасні веб-сторінки переважно розроблюються з використанням HTML+CSS+DOM [9], що і буде розглянуто у роботі.

Задля підтримки попередніх версій цих технологій, кожна наступна має зворотню підтримку. Таким чином маємо досить токсичне середовище розробки з такими проблемами як:

- наявність декількох методів для виконання однакових задач;
- погана структуризація API [14];
- нелогічність поведінки деяких API, що пояснюється словом “historical” на рівні специфікацій;
- направлення розробки переважно керується примхами наддинамічного ринку (особливо його найбільших гравців), а не здоровим глуздом.

Це особливо помітно при вивченні використання вбудованої мови програмування JavaScript для керування DOM.

Рішенням (з іншого боку, частиною проблеми) можна вважати цілий зоопарк з мов-надбудов над CSS та JavaScript, сторонніх бібліотек, та програмних платформ розробки інтерфейсів, де велику популярність мають реактивні компонентно-орієнтовані програмні платформи.

Серед популярних [4] реактивних програмних платформ можна виділити Vue.js [13, 38], Angular, React. Вони у тому чи іншому вигляді можуть забезпечити:

– розділення всього інтерфейсу на незалежні компоненти;

– спрощення роботи зі станом даних (state) додатку завдяки реактивній парадигмі програмування та модулям керуванням станів (state managers);

– декларативного підходу при розробці шаблонів форм, що мінімізує пряме використання DOM API через JavaScript;

– різноманітні оптимізації щодо зменшення перемалювання сторінки при зміні стану (Virtual/Shadow DOM [17], та інше);

– зміною форм без повного перезавантаження сторінок – створення так званих Single Page Applications.

Завдяки значної популярності, до цих програмних платформ можна знайти якісну і змістовну документацію, приклади та набори готових компонентів для значного спектру ситуацій, причому безкоштовно.

Таким чином, для створення динамічного веб-додатку використання таких програмних платформ особливо рекомендовано. Це дозволяє у короткий строк створити інтерактивний, швидкий та надійний сучасний веб-додаток.

1.3.2 Протокол передачі повідомлень для чата

Чат передбачає обмін повідомленнями (зазвичай короткими текстовими) у реальному часі з мінімальними затримками. У будь-якому якому випадку, чат потребує підтримки постійного з’єднання для того, щоб не пропустити нове повідомлення як зі сторони клієнта, так і зі сторони сервера.

Для забезпечення такого обміну часто використовують два основних підходи – метод Long-polling та протокол Websockets [36].

1.3.2.1 Метод Long-polling

Плюсом використання методу Long-polling [18, 39] є факт використання звичайного HTTP протоколу, тобто метод має широку підтримку як зі сторони серверів, так і веб-браузерів. Алгоритм виконання Long-polling наступний:

а) клієнт посилає запит до сервера;

б) сервер не закриває з’єднання доки не має повідомлення для відправлення, відправляє його;

в) клієнт отримує повідомлення, з'єднання закривається;

г) клієнт відразу створює нове з’єднання та цикл повторюється.

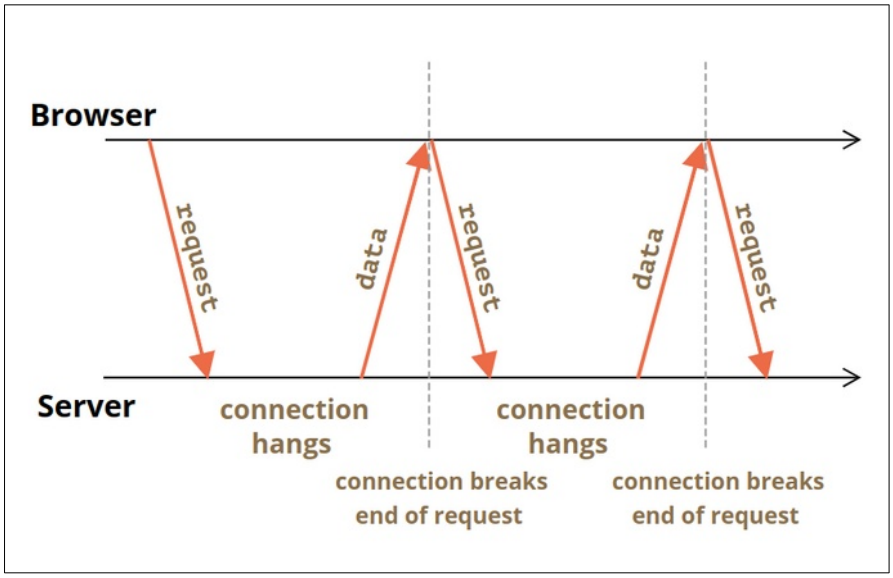


Рисунок 1.1 – Діаграма запитів при Long-polling

З основного плюса цього методу – використання звичайного протоколу HTTP – маємо й основні недоліки, як то:

– необхідність постійно створювати нові з’єднання – додаткова затримка у часі;

– необхідність постійно оброблювати Timeout помилки;

– постійна передача зайвої інформації – поля заголовків HTTP;

– запити є синхронними.

1.3.2.2 Протокол Websocket

Більш новий протокол Websocket вирішує (можна сказати, був створений щоб вирішити) недоліки методу Long-polling, маючи лише один значний недолік – факт новизни, проте на момент 2020 року усі сучасні веб-браузери та багато веб-серверів мають гарну підтримку цієї технології.

Websocket будується поверх протоколу транспортного інтернет протоколу, як TCP, він створює постійне двостороннє з’єднання. Обмін заголовками проходить лише при початковому Handshake з’єднанні.

Протокол обмінюється пакетами, що можна вважати значним недоліком, коли мова йде про використання його для передачі значного об’єму даних, але для подальшої розробки є найкращим варіантом. Наступним кроком у еволюції веб-API цього типу можна вважати протокол WebTransport [11], проте він наразі у стані чорнового нарису стандарту відповідної групи IETF [15].

1.3.3 Асинхронні веб-сервера

1.3.3.1 Синхронний, асинхронний та комбінований підхід обробки запитів

Однією з головних проблем, що має вирішувати веб-сервер є проблема багатьох одночасних запитів. Кожний такий запит містить:

- очікування та прийом даних з сокета від клієнта;
- деяку обробка запита з повним завантаженням ЦП;
- майже завжди, виконання запиту до БД або запис даних на диск;
- відправку та очікування завершення відправки даних на сокет клієнту.

Зазвичай, більша частина обробки запитів пов’язана саме з операціями зчитування / запису даних до бази даних або на диск (або виконання

додаткового запиту) та прийому від і відправки даних до клієнта. Вочевидь, під час цих операцій ЦП фактично не завантажений і міг би виконувати іншу роботу.

Класичний підхід для обробки багатьох запитів веб-сервером (на прикладі Apache [37]) заключється у створенні окремого потоку на рівні ОС для кожного з'єднання. Вочевидь, при такому підході отримується досить невелика максимальна кількість одночасно оброблюваних з'єднань, так як операція створення потоку досить затратна по пам'яті. До того ж, додатковий час займає перемикання між потоками задля забезпечення багатозадачності.

Цю проблему вирішує асинхронний підхід [44]. При ньому, коли настає момент виконання операції зчитування/запису, оброблююча функція "заморожується", віддаючи виконання назад до черги виконання. Ця черга сортується за статусом готовності операції зчитування/запису (надалі скорочено просто з/з) і виконує наступну за цим пріоритетом функцію. Таким чином, центральний процесор отримує максимальне можливе завантаження не витрачаючи часу на очікування завершення файлової операції. Все це виконується у один потік.

Програма (цикл виконання) дізнається про готовність операцій з/з завдяки відповідних API операційних систем, що викликають callback циклу виконання, коли потрібна операція завершена. У Linux цей API називається epoll [3, 48].

Можна помітити, що асинхронний підхід не використовує усю потужність багатоядерних процесорів. Це вирішується комбінуванням підходів – запуск по одному асинхронному процесу на кожний процесор.

Беручи до уваги, що сервіс розроблюється з врахуванням потенційної великої кількості одночасних запитів, а також для роботи з websocket протоколом, що розуміє під собою велику кількість з'єднань у режимі очікування, комбінований підхід є найкращим рішенням.

1.3.3.2 Обрання програми сервера

Існує величезна кількість різноманітних реалізацій асинхронних програм серверів (надалі просто сервер). Слід також зазначити, що часто мова йде про деякий програмний застосунок поверх сервера, що надає необхідний стандартний функціонал, як то робота з cookie, http-заголовками, маршрутизація, та інше. Можна вивести наступні критерії пошуку оптимального веб-серверу:

- мова програмування;
- простота та швидкість програмування взагалі;
- модульність, орієнтованість на просте створення JSON API [16] або навіть REST API [19];
- якість документації та прикладів;
- загальна якість коду, рівень покриття тестами;
- активна спільнота розробників;
- наявність готових необхідних для розроблюваного сервісу компонентів та функціоналу (як то Websocket, JWT, авторизація користувачів та інше);
- відносна швидкість обробки запитів та об’єми використання пам’яті.

Можна помітити, що більшість, або навіть всі (так як навіть питання вимірювання продуктивності фактично складно формалізувати) з цих критерієві є значною мірою суб’єктивні.

Автор найкраще знайомий з “екосистемою” розробки на базі мови Python [28] та програмна платформа Flask [40] на базі веб-сервера Werkzeug (WSGI), що не є асинхронним, проте відповідає більшості з зазначених критеріїв. Таким чином, вибір було швидко зведено до використання відносно нового асинхронного програмної платформи fastAPI на базі асинхронного серверу Starlette (ASGI).

1.3.3.3 Програмна платформа для веб fastAPI

FastApi [12] – це досить нова програмна платформа асинхронного веб-сервера на базі Starlette [31]. Ключові особливості:

а) гарна підтримка IDE завдяки анотаціям типів програмного коду;

б) підтримка інструментарію для створення JSON Web API з коробки, на приклад:

1) обробка JSON запитів та відповідей за замовчуванням (націленість на сумісність зі специфікацією OpenAPI [25]);

2) використання анотацій типів та моделей Pydantic [27] (у тому числі і вкладених моделей) як засобу декларації вхідних даних (десеріалізації) а також серіалізації відповідей, що додатково видає ємні та структуровані відповіді при помилках;

3) автоматичне генерування документації;

в) підтримка middleware;

г) загальна простота використання, ясність коду.

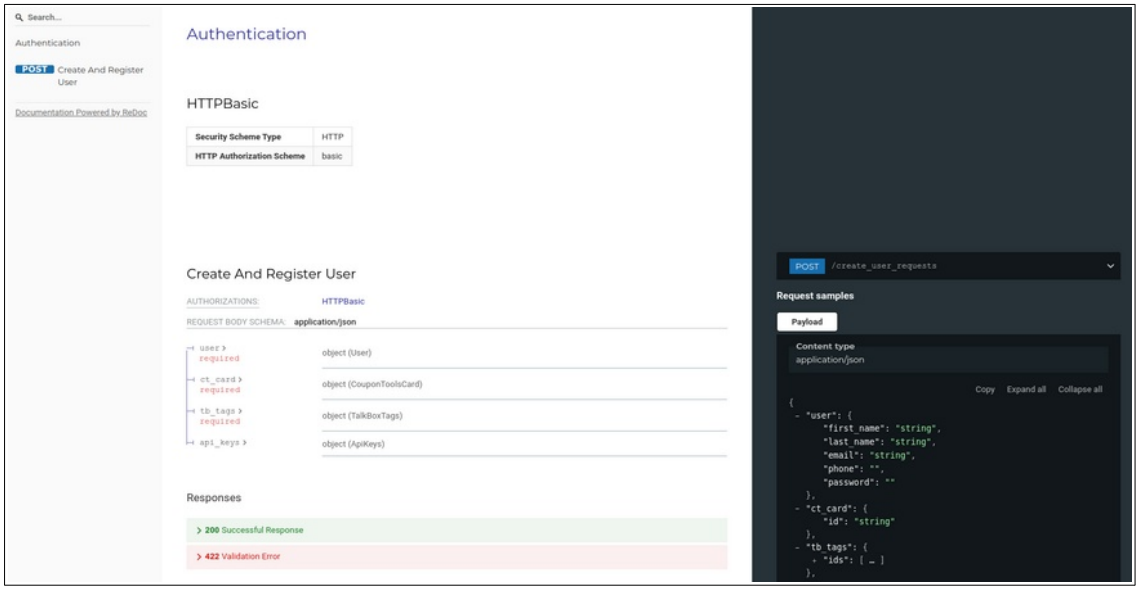


Рисунок 1.2 – Автоматично згенерована документація fastAPI – Redoc

Також висока якість документації як самої програмної платформи, так і серверу-базування Starlette, модулю валідації даних Pydantic.

1.3.4 Вибір між БД типу SQL та NoSQL

На сьогоднішній день питання SQL vs NoSQL [29] має високу популярність. Спершу, слід зазначити, що під терміном SQL мається на увазі перш за все реляційні бази даних, а не конкретне використання БД мови SQL; тоді як термін NoSQL іноді розширюють до Not Only SQL, але в цілому суть одна – відмінні від реляційних баз даних, такі як індексні, графові, документі, словникові (ключ-значення), та інші.

Головна проблема цього питання у типу масштабуванні: звичайні SQL БД масштабуються вертикально (проте цю проблему намагаються вирішити засобами так званих NewSQL та Distributed SQL), тоді як сучасні NoSQL створювались з розрахунком на горизонтальне масштабування. Додатково, більшість NoSQL БД не потребують строгого задання схеми (структури) даних, що особливо актуально для динамічно розвиваємих проектів.

Горизонтальне масштабування та нестрогість схеми даних та відношень породжують свої окремі проблеми, особливо щодо питання розподілених систем, що відображає CAP теорема (або теорема Брюера) [6]: можливо обрати лише 2 характеристики з трьох – узгодженість, доступність, стійкість до поділу.

Враховуючи специфіку розроблюваного додатку (це не є критична система реального часу, як то банківська, медична, або керування апаратним забезпеченням деякого підприємства), що націлений на горизонтальне масштабування, та бажання мати додаткову гнучкість та зниження складності розробки, основною БД було обрано об'єктний тип БД CP типу за CAP теоремою, а саме MongoDB.

1.3.4.1 MongoDB

Відповідно до попереднього пункту, MongoDB – це документна база даних [22], що не потребує вказання схеми даних. Перша реліз-версія датується 2009 роком, є найпопулярнішою [23] серед NoSQL баз даних, отже проблем з надійністю, документацією та використанням API не слід очікувати.

MongoDB реалізує засоби забезпечення цілісності даних завдяки створенню кластера з декількох повних копій БД (Replica set), причому лише один екземпляр процесу БД має право на запис.

Для забезпечення горизонтального масштабування (у тому числі на рівні файлової системи) MongoBD реалізує концепт Sharding.

MongoDB дозволяє створювати індекси по будь-якому полю, має свій багатий синтаксис запитів. Надійність досягається тим, що серед набору реплік БД може бути лише один головний (тобто маючий право на запис).

Можна провести аналогію термінів відповідно до реляційних баз даних, як зазначено нижче.

Таблиця 1.1 – Відповідність термінів РСКБД до MongoDB

РСКБД	**MongoDB**
База даних	База даних
Таблиця	Колекція
Кортеж / запис	Документ
Стовпець	Поле
З’єднання таблиць	Вкладений документ

Особливо спрощує роботу концепція вкладених документів, приклад чого можна бачити на рисунку нижче.

```
// patron document
{
   _id: "joe",
   name: "Joe Bookreader"
}

// address documents
{
   patron_id: "joe", // reference to patron document
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}

{
   patron_id: "joe",
   street: "1 Some Other Street",
   city: "Boston",
   state: "MA",
   zip: "12345"
}
```

```
{
   "_id": "joe",
   "name": "Joe Bookreader",
   "addresses": [
                 {
                   "street": "123 Fake Street",
                   "city": "Faketon",
                   "state": "MA",
                   "zip": "12345"
                 },
                 {
                   "street": "1 Some Other Street",
                   "city": "Boston",
                   "state": "MA",
                   "zip": "12345"
                 }
               ]
}
```

Рисунок 1.3 – Приклад об’єднання нормалізованих сутностей до одного документа [21]

1.3.5 Контейнеризація засобами Docker

Розгортання програмного забезпечення (навіть на машині іншого розробника), особливо серверного, часто потребує збірку та встановлення необхідних залежностей, компіляторів чи інтерпретаторів, виконання різноманітних налаштувань. Причому, на різних середовищах цей процес може проходити по різному.

Цю проблему вирішує контейнеризація ПЗ – пакування усіх залежностей та внутрішніх налаштувань у один виконуємий файл – контейтер.

Слід зазначити 2 основні підходи контейнеризації: використання віртуальних машин (VMWare, VirtualBOX) та легких контейнерів (Docker, Podman [42, 45]). Головна різниця [8] в тому, що повна віртуалізація потребує гостьової ОС, тоді як легкі контейнери роблять виклики напряму до ядра приймальної (host) ОС, що зображено на рисунку нижче.

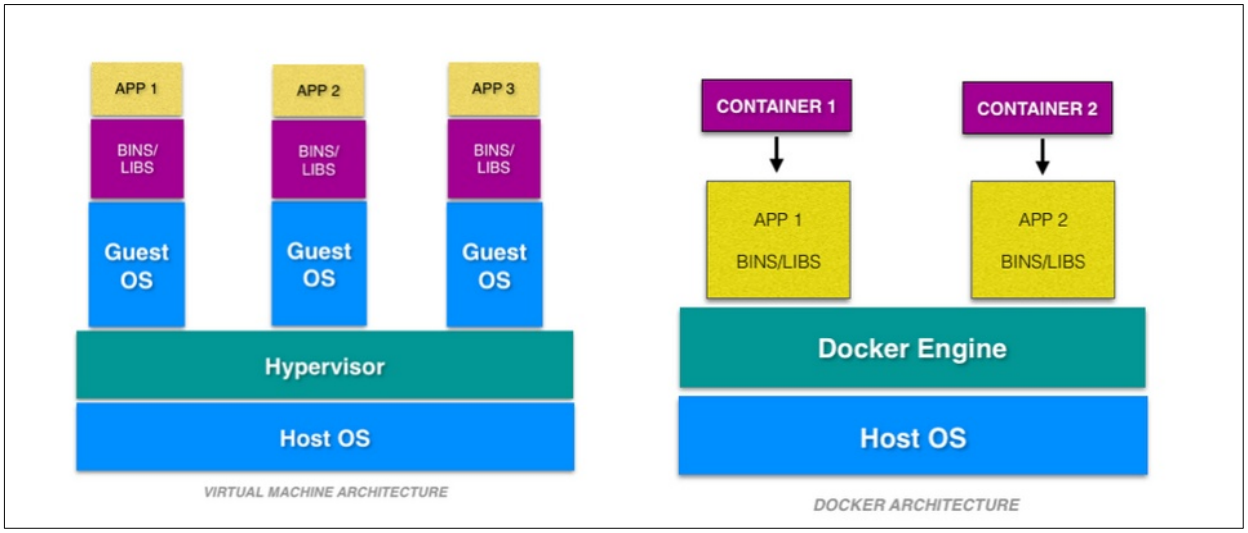


Рисунок 1.4 – Архітектурна схема віртуальних машин та Docker

Docker [45] наразі фактично є стандартною технологією контейнеризації, що підтримують усі головні гравці ринку хмарних технологій. Так як контейнери не потребують гостьової ОС і працюють майже напряму з ядром приймальної ОС, вони є малими за використовуваним дисковим об’ємом та швидкими як за розгортанням, так і за роботою в цілому.

Крім того, контейнери Docker можуть напряму виступати у ролі одиниці оркестрації контейнерів Kubernetes, [41] про що описано у наступному підрозділі.

1.4 Керування контейнерами засобами Kubernetes

Наступним кроком після отримання набору контейнерів є передача їх до системи керування контейнерами (поширений термін – система оркестрації), що здійснюватиме процес автоматичного розгортання, масштабування та в цілому керування контейнерами. Доцільності такої системи розкриває наступна гіпотетична ситуація: потрібно розгорнути 200 контейнерів на 100 машинах, балансувати навантаження між вузлами кластеру, перезавантажувати контейнери при оновленні програмного

застосунку, відслідковувати статус роботи та перезавантажувати неробочі контейнери та інше, а також відслідковувати об’єми вжитих ресурсів для постачальників інфраструктури для проведення обчислення.

Фактичним стандартом у індустрії виступає система з відкритим вихідним кодом Kubernetes, початково розроблена Google. Своє налаштування середовища Kubernetes дозволяють виконувати поміж усіх інших такі постачальники хмарних обчислень як Google, Amazon та Microsoft [2, 10].

На найнижчому програмному програмному рівні Kubernets оперує контейнерами. Базова ж одиниця керування програмними додатками є так званий Pod [26] – сукупність однакових контейнерів (один або більше), що є об’єктами балансувальника навантаження, звертаючись по віртуальному IP до одного Pod фактично виконується звертання до одного з декількох екземплярів специфічного контейнеру за рішенням балансувальника навантаження.

Для спілкування з зовнішнім світом, Pods мають бути пов’язані зі спеціальним Proxy сервісом.

Kubernetes Master контролює роботу усієї системи розподіляючи навантаження по окремим вузлам – фізичним чи віртуальним машинам, та надаючи засоби керування оператору кластера. Загальна схема зображена на рисунку нижче.

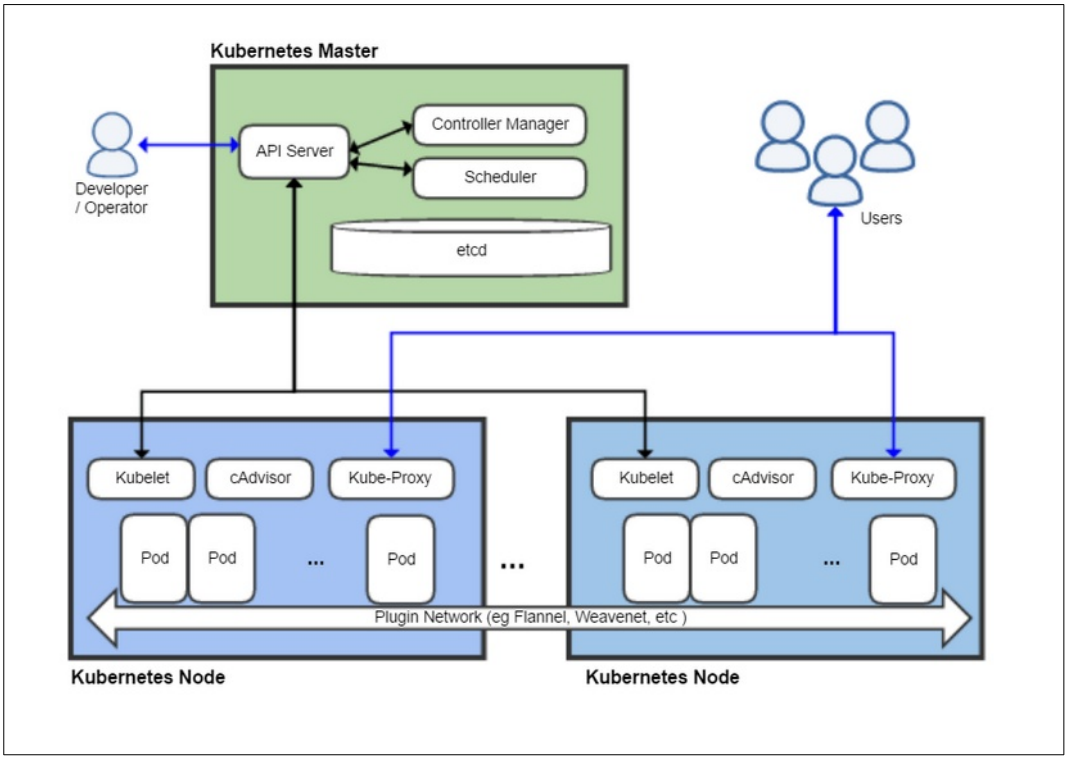


Рисунок 1.5 – Архітектурна схема системи Kubernetes

Налаштування за замовчуванням зберігаються у декларативному форматі у YAML [46] файлах, а для доступу до окремих вузлів системи використовуються спеціальні селектори.

1.5 Висновки за розділом

У даному розділі було розглянуто деякі з факторів мотивації у виконанні задач, а також дано визначення поняттям задачі та цілей. Маючи на увазі ці поняття та фактори, було виведено перелік критеріїв для порівняння існуючих сервісів організації особових задач, як тих, що лише реалізують аспект структурної організації задач, так і тих, що застосовують у тому чи іншому вигляді соціальний фактор.

Відповідно до визначених критеріїв було проведено порівняння сервісів, та виведено перелік вимог, які, на думку автора, відповідали би сервісу наступного покоління.

У подальших підрозділах було проведено огляд головних компонентів та програмних платформ і бібліотек для реалізації сучасного веб-застосунку як зі сторони клієнтського коду, так і зі сторони сервера, що забезпечить одночасну обробку великої кількості запитів завдяки масштабуванню на рівні інфраструктури та більш ефективного використання процесорного часу (асинхронні сервера).

РОЗДІЛ 2
РОЗГОРНУТИЙ КОНЦЕПТ ПРОЕКТУ РОЗРОБЛЮВАНОГО СЕРВІСУ

Цей розділ має на меті на основі матеріалів з попереднього розділу сформувати бачення розроблюваної системи, деякий матеріал, на основі котрого можна буде висовувати кінцевий перелік задач на розробку.

Слід зазначити, що було вирішено перейти до концепції більш вузької специфікації сервісу, на відміну від напрямку на універсифікацію, як зазначалося у попередньому розділі. Фокус роботи зміщено у сторону взаємодії користувачів лише на базі задач, тобто по завершенню задач подальше спілкування учасників кооперації вважається зайвим. Додатково, користувацький інтерфейс також обрано створювати у мінімалістичному стилі з мінімальною кількістю переходів між цільовими діями.

У термінології розроблюваного сервісу елементами кооперації є проекти – набори задач.

2.1 Основні логічно-функціональні компоненти

Розбиття системи було проведено відповідно до основних компонентів користувальницького інтерфейсу та його функціоналу.

Процес такого розбиття суто творчий, тож можливе порушення чіткості кордонів перелічених далі компонентів, як то чат і система повідомлень, пошук проектів інших користувачів і система автоматичного підбору. Це вбачається допустимим, адже головною метою такого розбиття є отримання найбільш детального бачення кінцевого продукту, що у подальшому дозволить створити конкретний набір завдань на розробку, де і буде проведена кінцева деталізація вимог до розроблюваної системи.

Таким чином було виділено наступні компоненти:

– реєстрація та авторизація;

- особистий профіль користувача;
- особистий перелік розділів, проектів та задач;
- пошук проектів інших користувачів для кооперації;
- чат;
- система повідомлень;
- система автоматичного підбору користувачів;
- календар задач.

Кожний такий компонент матиме опис необхідного функціоналу та ескізи форм (де необхідно).

2.1.1 Реєстрація та авторизація

Цей компонент відповідатиме за досить стандартну форму реєстрації та авторизації з головної сторінки. Додатково, ця головна сторінка виконуватиме роль пояснювальної сторінки, тобто описуватиме про що є цей сервіс.

Таким чином, потрібно реалізувати наступні компоненти:

а) початкова сторінка-привітання з формою реєстрації, авторизації та форми запиту на відновлення паролю (з перемиканням між цими компонентами без перезавантаження сторінки);

б) кнопку авторизації (реєстрації) з використанням OAuth протоколу;

в) відповідні шаблони поштових повідомлень.

Форма авторизації вимагатиме пошту та пароль.

Форма реєстрації вимагатиме наступні поля:

- ім’я
- прізвище
- пошту
- пароль та його підтвердження

У процесі реєстрації у користувача буде запитано лише найнеобхідніші поля, таким чином зменшуючи кількість тих, хто не дійшов до кінця цього процесу.

У разі успішної реєстрації буде відображено сторінку з повідомленням про успіх операції та запрошенням перейти до пошти задля активації облікового запису. Для додаткового комфорту, буде відображено посилання на поштовий домен, тобто якщо користувач увів пошту “vasil@gmail.com”, сторінка підтвердження буде мати посилання на “gmail.com”.

Лист з підтвердженням буде мати посилання на сторінку, що відобразить повідомлення про успіх операції підтвердження (або помилку з рекомендацією звернутися до адміністрації за додатково вказаним посиланням). У свою чергу, сторінка підтвердження матиме посилання з запрошенням до редагування особистого профілю для заповнення додаткової інформації.

Форма запиту на відновлення паролю вимагатиме лише пошту. Задля підвищення безпеки, форма не буде повідомляти чи існує користувач за вказаною поштою.

2.1.2 Особистий профіль користувача

Компонент особистої публічної сторінки користувача складатиметься з двох компонентів – компонент відображення та редагування.

Матимемо наступні інформаційні редагуємі поля профілю:

- ім’я;
- дата народження;
- стать;
- країна проживання;
- вивчені мови та рівень їх оволодіння (chatting, fluent, native);
- часовий пояс;

– аватарка;

– інформація о собі (з використанням Markdown).

Слід зазначити, що режим відображення замість дати народження відображатиме лише поточний вік користувача, а до часового поясу буде додана різниця між вказаним часовим поясом та власним часовим поясом, якщо переглядається чужа сторінка профілю.

У майбутньому, блок інформації о собі можливо буде розділити на деяку структуру щодо деяких особливостей та вподобань користувачів задля як збільшення інформативності профілю, так і для покращення роботи алгоритмів пошуку рекомендованих завдань для кооперації, про що буде у відповідному розділі.

Сторінка профілю також відображатиме статистику щодо виконання завдань у вигляді суми балів за увесь час та графіку набраних балів у проміжках неділь та місяців разом з максимально можливими балами, таким чином відображаючи якість виконання завдань.

Останнім блоком (хоча можливо є доцільним відображати його одним з перших) є блок проектів з відкритим місцем на кооперацію. Кожний проект з цього блоку буде відповідати відображенню проектів з блоку пошуку проектів на кооперацію – дивитися у відповідному підрозділі.

Схематичний вигляд форми відображення зображено на рисунку 2.1 нижче.

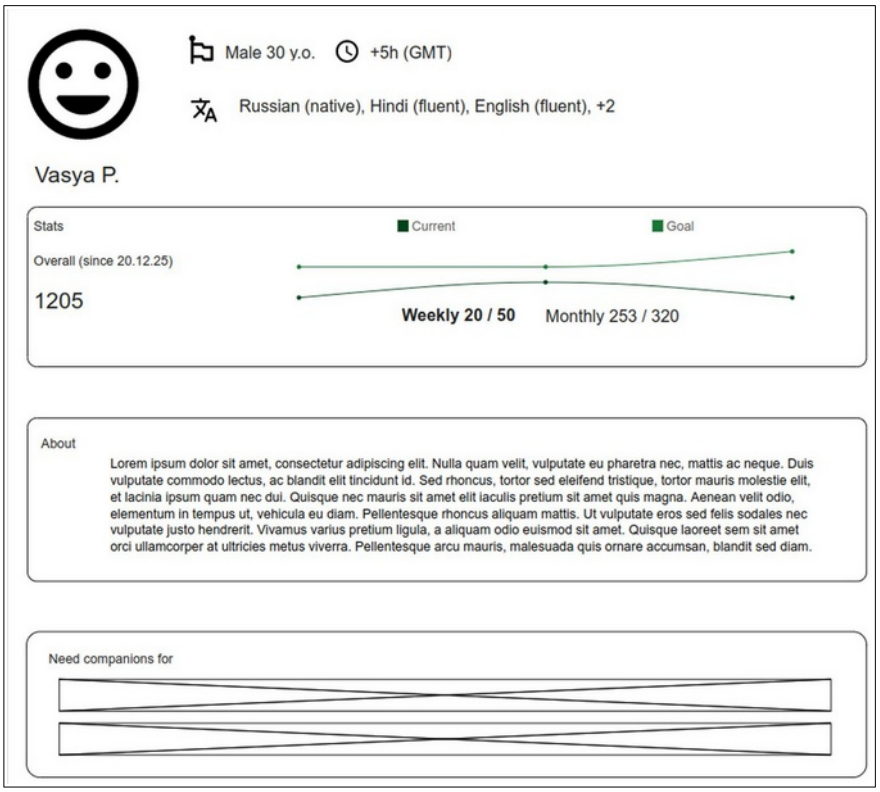


Рисунок 2.1 – Концептуальна схема форми профілю користувача у режимі відображення

2.1.3 Особистий перелік розділів, проектів та задач

Задля спрощення, цей розділ не буде функціонувати як місце для відмічання фактичного прогресу, а лише для редагування та відображання проектів та інших структур. Відмічанню прогресу слугуватиме компонент календаря.

Як і у попередньому підрозділі, цей компонент буде реалізовувати 2 режими: режим відображення і редагування (додавання) елементів. Задля спрощення, форма редагування буде реалізована не на місці, а у вигляді pop-up вікна.

Структура елементів виконання наступна: розділ -> проект -> задача. Слід зазначити, що об’єктом кооперації є саме елемент проекту.

2.1.3.1 Елемент розділу

Розділ слугуватиме для додаткового рівня структуризації особистих завдань користувача. Існує немало різних популярних видів розподілу завдань (наприклад, за сферами життя, як то робота, навчання, здоров’я та інше). На момент написання було обрано дозволити довільні розділи, особливо маючи на увазі, що розділи самі по собі не розглядаються як елементи кооперацій, в інакшому випадку введення стандартизованого набору розділу могло б допомогти у збільшенні варіантів пар для кооперації.

Розділ матиме статистику агреговану з включених у нього проектів. Розділ матиме 2 основних поля: назва та теги. Теги розділу стають тегами за замовчуванням (але не обов’язковими) для проектів.

2.1.3.2 Елемент проекту

Проект, додатково до полів назви, тегів та агрегованої статистики (вже по включеним задачам), отримує поле дати початку та кінця, що лімітують відповідні поля вкладених задач, а також короткого опису. Проект можна переміщювати між розділами, проте, задля спрощення як реалізації так і логіки користування, проект не може мати більше ніж один розділ.

Проект є об’єктом кооперації, тож повинен мати відповідні поля відображення та редагування. Також, проект матиме посилання на відкриття відповідної кімнати у чаті.

Концепт дизайну форми проекту наведено на рисунку нижче. При наведенні на знак запитання буде відображено “хмарку” з поясненнями.

Слід зазначити, що на тому рисунку не відображено налаштування для приватності, що може бути у стані “публічний”, або “приватний”. У випадку, коли обрано приватний стан проекту, він не буде включений до результатів пошуку проектів на кооперацію, а відповідно не буде включено й до розгляду системою автоматичного підбору компаньйонів.

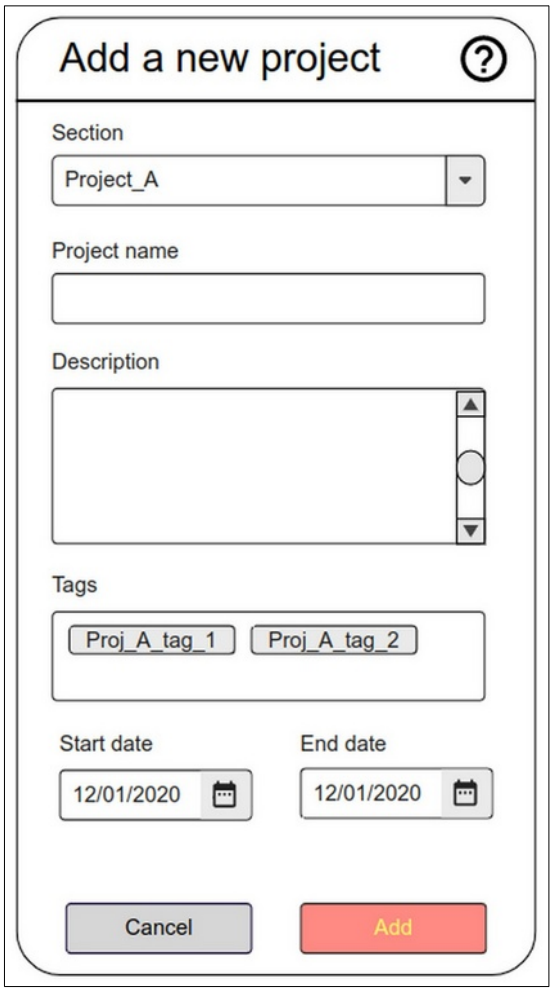


Рисунок 2.2 – Концепт форми додавання проекту

2.1.3.3 Елемент задачі

Задача – головний елемент процесу виконання особистих завдань, він матиме контролери для відмічання прогресу. Задача не матиме поля тегів та не може бути переміщеною до іншого проекту.

Контролер відмічання налаштовується відповідним полем відмічання, що нормалізується відповідно до очікуваного результату та зберігається у вигляді дійсного числа. Це поле може бути наступних типів:

– одиниці, що супроводжуються назвою одиниці вимірювання;
– відсотки;

– тип часу, що записується у позиційному вигляді, тобто “години : хвилини” (якщо надано лише одне значення, то воно сприймається за хвилини).

Іншою відмінністю задач від проектів є додаткове поле конкретного часу виконання та налаштування повторюваності задачі. Замість кнопок та переліків для завдання повторюваності, можливо (при достатньому часі на розробку) використовувати строковий формат завдання повторень, як то «Every second day since 10 Jan 2019 till 15 Nov 2020» – аналогічна система з Todoist.com.

Додатково, відмічання задачі породжуватиме елемент системи події, що буде вписано у відповідний чат кооперації.

Концепт дизайну форми додавання задачі наведено на рисунку нижче.

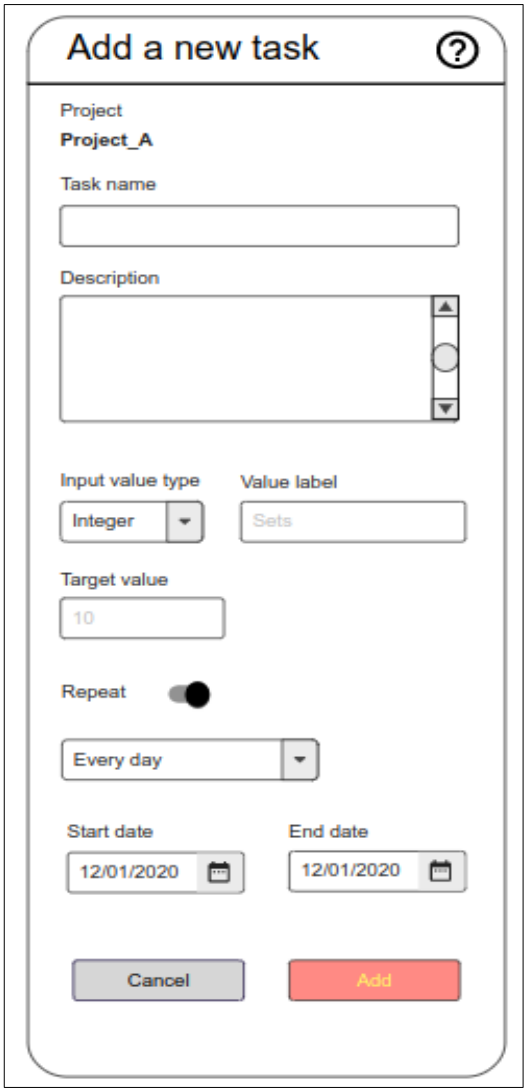


Рисунок 2.3 – Концепт форми додавання задачі

2.1.4 Пошук проектів інших користувачів для кооперації

Для ручного пошуку та перегляду усіх можливих проектів для кооперацій необхідно мати окремий компонент пошуку у вигляді окремої сторінки.

Сторінка пошуку складатиметься з двох основних компонентів:

а) компонент налаштування фільтру та кнопки відправлення запиту;

б) компонент відображення результатів пошуку.

2.1.4.1 Компонент налаштування фільтру

Компонент складається з двох елементів – поле уведення запиту та кнопки відправлення.

Запит матиме специфічний синтаксис, а саме:

а) для пошуку по тегам потрібно додавати префікс “#”, наприклад “#диплом”;

б) для пошуку за ім’ям або унікальним ідентифікатором користувача слід додати префікс “@”, причому для двоскладового ім’я (ім’я та прізвище), слід включити слова запиту у лапки, наприклад “@"Василь Пупкін"”;

в) для пошуку за назвою або описом достатньо просто ввести ключові слова як є.

У подальшому подібний підхід можна розширювати і для інших типів ключових слів, проте при занадто великій їх різноманітності слід перейти до більш класичної форми налаштування – зі списками та radio-button.

2.1.4.2 Блок результатів та елемент проекту

Блок результатів складається з окремих компонентів проектів а також, при великій кількості результатів, блоку перегортання сторінок (pagination).

Критерій фільтрації наступний:

$$\left\langle \frac{fT}{tT} * cT \right\rangle * \left\langle max(\mathrm{LL}) * cL \right\rangle * \left\langle \frac{fD}{tD} * cD \right\rangle \qquad (2.1)$$

де fT – фактична кількість співпалих тегів,

tT – загальна кількість тегів,

cT – коефіцієнт важливості критерію тегів,

LL – рівень оволодіння спільною мовою у діапазоні [0..3],

cL – коефіцієнт важливості критерію мови,

fD – кількість збіглих слів з опису та назви,

tD – загальна кількість слів у описі та назви,

cD – коефіцієнт важливості критерію співпадіння опису та назви.

Вочевидь, що така формула має багато недоліків, але як для першої версії сервісу можна вважати її достатньою. У майбутньому слід розглянуті наступні аспекти поліпшення:

– вказувати користувачам вводити теги у порядку конкретизації елементів (розділів та проектів), після чого підібрати більш ефективний алгоритм роботи саме з такою структурою;

– не “створювати велосипед”, а використати готові бібліотеки повнотекстового пошуку (якщо такі наявні для використовуваної БД та мови програмування сервера) зі сортуванням за релевантністю.

Результати відображаються відсортованими за цим критерієм. Коефіцієнти важливості окремих факторів будуть підібрані під час випробувань у реальній роботі.

На рисунку нижче можна бачити концепт вигляду форми пошуку. Як можна бачити, елемент результату складається з двох головних компонентів – елементу відображення автора проекту та інформації про сам проект.

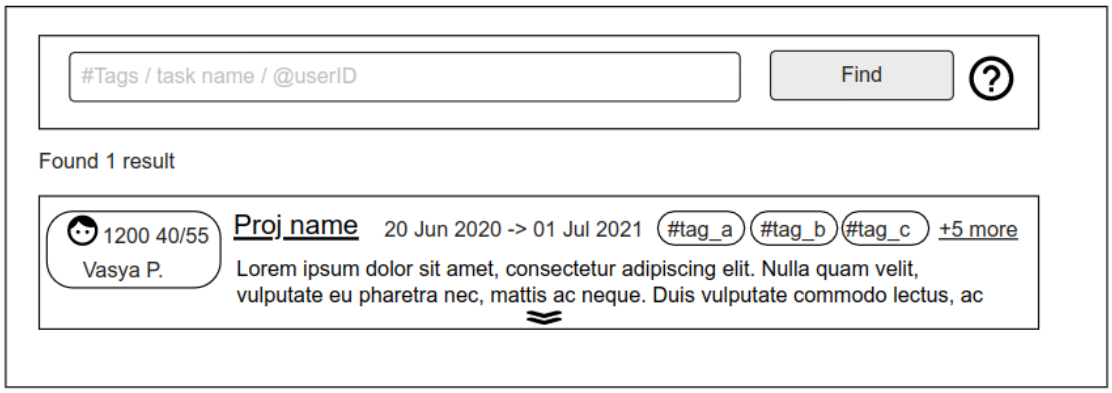


Рисунок 2.4 – Концептуальний дизайн форми пошуку проектів для кооперації

Елемент автора проекту по суті є посиланням на персональну сторінку відповідного автора. Елемент відображає сумарну кількість балів та бали за останній тиждень (фактичні та цільові).

Елемент опису проекту складатиметься з:

а) назви проекту, що буде посиланням на відкриття розгорнутої форми проекту з елементами для ініціювання процесу кооперації;

б) дати початку та завершення проекту;

в) тегів, що можна по натисканню додати до поточного поля пошуку;

г) опису, що по натисканню на відповідну іконку буде розгорнуто, якщо він займатиме більше відведеного за замовчуванням місця.

2.1.5 Чат

Чат є головним фактором підтримки кооперації шляхом як, власне, спілкування, так і завдяки виконання ролі форми відображення системи повідомлень.

Особливістю чата розроблюваного сервісу мають стати так звані “спеціальні чат-об’єкти” – включення до текстових повідомлень, що мають та мають змогу відображати додаткову мета-інформацію. Наприклад, спеціальний чат-об’єкт повідомлення про виконання завдання за замовчуванням виглядатиме як деким чином виділений текст, при наведенні – відобразить повний текст коментаря до виконання, а при натисканні – відкриє календар у відповідному цій задачі місці. Іншими спеціальними об’єктами планується мати: повідомлення про зміну плану та цитати (з можливістю переходу до місця цитування у історії розмови).

Чат поділяється на кімнати для розмов (головне меню чата). Відокремлено має бути кімната агрегації повідомлень – усі повідомлення будуть збиратися до цієї кімнати, проте ці повідомлення також будуть збережені у відповідних кімнатах кооперації. При натисканні на таке повідомлення відкриється відповідна кімната чата з додаванням до поля

уведення повідомлень як спеціального об’єкту, таким чином дозволяючи швидко перейти до обговорення події.

В цілому, головне меню чата відображатиме строки кімнат з елементами налаштувань щодо повідомлень, аватаркою-посиланням на профіль відповідного компаньйона, статусу онлайн компаньйона та кількості нових повідомлень. І на останок, елементом-кнопкою закриття чата форми. Концепт зображено на рисунку 2.5.

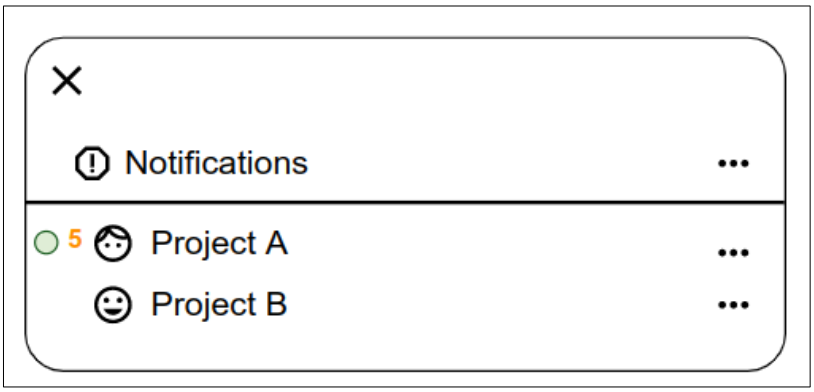


Рисунок 2.5 – Концептуальний ескіз дизайну форми меню чата

Сама кімната чата складатиметься з трьох основних компонентів: компоненту введення нового повідомлення, компоненту історії повідомлень та навігаційного компоненту. Загальна концепція дизайну відповідає тому, що можна бачити у сучасних додатках чатів для смартфонів.

Знизу – елементи уведення повідомлення: текстове поле, кнопки смаликів та прикріплення файлів (головним чином очікується використання цієї функції для додавання зображень, проте її можна також реалізувати при зчитуванні буферу обміну при CTRL+V).

Самі елементи повідомлень будуть зсунуті у сторону, таким чином надаючи додаткову візуальну підказку про автора повідомлення (особливо актуально якщо розвивати ідею парної кооперації до групової), також власні повідомлення можна виділяти рамкою іншого кольору. Збоку від

повідомлення відображатиметься аватарка відправника та час відправлення повідомлення. Взагалі, при розробці чата з якнайбільш двома учасниками, може бути достатньо зміни кольору обрамлення повідомлення для відрізнення власного повідомлення, проте обраний підхід є загальноприйнятим шаблоном, отже не потребуватиме додаткового часу для вивчення.

Останнім елементом слугуватиме навігаційний блок зверху. Він матиме кнопки для закриття вікна чата, переходу до меню чата, посилання на сторінку власного проекту кооперації, посилання на сторінку компаньйона та його проекту. Також, у цьому компоненті буде розміщено кнопку перемикання відображення автоматично доданих повідомлень.
Концептуальний ескіз дизайну форми кімнати чата можна бачити на рисунку нижче.

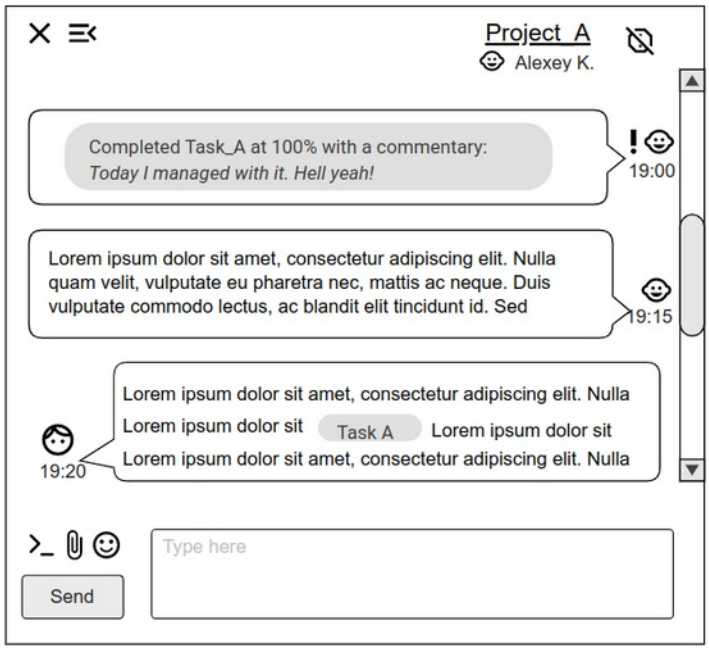


Рисунок 2.6 – Концептуальний ескіз дизайну форми кімнати чата

Останнім елементом чата є набір кнопок для відкривання чата. Вбачається, що вони будуть доступні на кожній сторінці у “висячому” (CSS

position:fixed) положенні з лівої або правої сторони сторінки на її кордоні. Ескіз форми наведено на рисунку нижче.

Рисунок 2.7 – Концептуальний ескіз дизайну форми кнопок відкриття чату

Цей елемент складатиметься з трьох компонентів:

а) кнопка та лічильник сповіщень, що по натисканню відкриє кімнату повідомлень;

б) кнопка та лічильник нових повідомлень, що відкриє кімнату з найпершим непрочитаним повідомленням;

в) кнопка та лічильник нових запитів на кооперацію, що працюватиме аналогічно попередній кнопці, проте не буде відображатися, коли таких запитів не має.

2.1.6 Система повідомлень

Повідомлення необхідні для підтримки активного залучення користувачів до робочого процесу кооперації. Система повідомлень пов’язує події у елементах проектів з їх відображенням у чаті. Додатково, ця система може надсилати повідомлення на пошту користувача (відповідні налаштування для спрощення вбачається помістити у налаштування кімнати сповіщень у чаті).

Як вже частково зазначалося раніше, повідомлення можуть бути згенеровані за наступних умов:

– прогрес у виконанні задач (їх відмічання, або автоматичне відмічання як невиконаних);

– будь-які змінах у проекті, як то опис, дати, теги, нові задачі, зміна існуючих задач, заміна батьківського розділу;

– поява нового запиту на кооперацію;

– поява нового повідомленнях у чаті.

Слід зазначити, що такі повідомлення не будуть відображатися явно провокуючим їх генерацію користувачам (тобто, автоматичне відмічання задачі як невиконаної не є явним провокуванням, тож таке повідомлення має бути відображеним).

Поміж чату, існує ще 2 способи відображення сповіщень: пошта та іконка у заголовку вкладки у браузері (надалі просто favicon.ico).

Сповіщення для посилання на пошту будуть за замовчуванням групуватися у один лист, що буде відправлено, наприклад, о шостій вечора, якщо до того моменту ще будуть непрочитані сповіщення. Ці налаштування будуть доступні у меню налаштування кімнати сповіщень, про що коротко буде згадано у самих листах сповіщень.

Додатково доцільно налаштовувати фільтрацію за рівнем важливості сповіщення, таким чином користувач не пропустить важливе сповіщення щодо нового запрошення на кооперацію серед однотипних сповіщень про виконання завдань або змін проектів. Таким чином можна виділити 3 рівня важливості:

а) нове повідомлення або запрошення на кооперацію;

б) зміни у планах;

в) прогрес по задачам.

Можливо виділити ще один тип повідомлень (другого рівня), а саме повідомлення щодо проблем з виконанням задач. Наприклад, коли користувач декілька днів поспіль виконує лише половину плану, слід надіслати сповіщення як самому користувачеві, так і його компаньйону, таким чином надавши додатковий стимул для початку обговорення.

Ці налаштування будуть змінювати поведінку для пошти та favicon.ico, але не блоку-індикатора нових сповіщень відкриття чата.

Звукові повідомлення та стимулюючи кольори до розробки не передбачаються, тим самим не експлуатуючи слабкістю людської природи, як це роблять популярні соціальні мережі.

2.1.7 Система автоматичного підбору користувачів

Очікується що великий відсоток нових користувачів (а також значний досвідчених) матиме проблеми з тим, щоб першими подати заявку на кооперацію. Цю проблему може частково вирішити система автоматичного створення заявок на базі системи пошуку відкритих на кооперацію проектів з попереднього підрозділу. Ця ідея (як і багато інших) взята з сервісу beta.getmotivatedbuddies.com.

Ця система буде автоматично запускатися за розкладом (наприклад, раз у 2 дні) і, використовуючи алгоритми з компоненту пошуку проектів на кооперацію, буде зіставляти найбільш схожі пари проектів (формула 2.1). Потрібно також задати нижній поріг рівня схожості, доцільне значення якого буде обрано на етапі тестування сервісу на користувачах.

Важливо, що користувачі матимуть лімітовану кількість часу (наприклад 48 годин) на прийняття або відхилення запиту. Користувачі (пара), що не здійснили жодного вибору будуть на деякий час заблоковані з системи автоматичного пошуку як неактивні користувачі. Додатково, користувачі з наявними необробленими заявками (у тому числі надісланими вручну), також будуть виключені з пошуку.

2.1.8 Календар задач

Компонент календаря призначений для відображення всіх (або відфільтрованих за проектами і / або розділами) завдань користувача та

компаньйонів за днями. Саме у календарі буде відбуватися процес відмічання виконання задач.

Формат відображення календаря вирішено робити за аналогією календаря з Todoist.com, що можна бачити на рисунку 2.8.

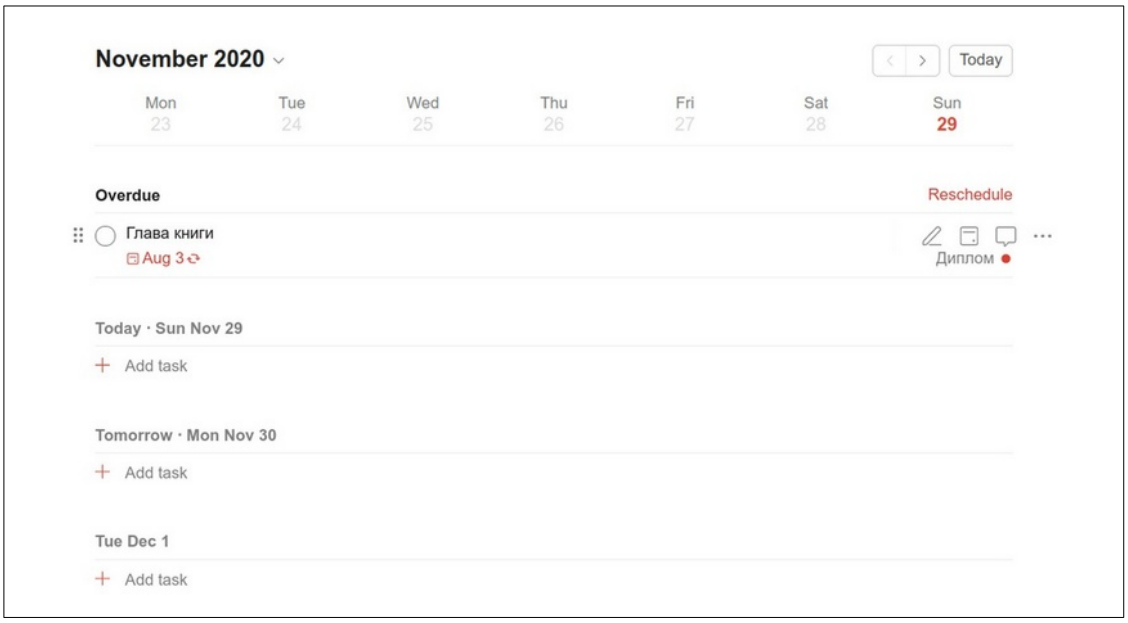


Рисунок 2.8 – Скриншот форми календаря Todoist.com

Календар матиме горизонтальну стрічку навігації за днями з швидким перемиканням між днями неділі. Самі ж задачі відображатимуться у вертикальному потоці займаючи усю ширину сторінки, таким чином досягаючи оптимального використання простору з максимізацією кількості корисної інформації та органів керування.

Додатково, календар матиме налаштування для фільтрації задач за проектами та розділами, фільтрацію задач компаньйонів, а також перемикання відображення вже виконаних задач.

2.2 Висновки за розділом

На основі загальних вимог до розроблюваного сервісу з першого розділу, у другому розділі було наведено детальний концепт розроблюваного

сервісу. Концепт поділено на набір функціонально-логічних компонентів, деякі з яких поділені далі. Наведено опис роботи функцій, а також макети форм. Таким чином забезпечуючи вхідний матеріал для простого створення переліку конкретних задач на розробку.

РОЗДІЛ 3
АРХІТЕКТУРА І ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ СЕРВІСУ

У цьому розділі буде описано архітектурні рішення у контексті деяких (важливих на думку автора) проблем у подальшій реалізації. Крім того, цей розділ описує сервіс як немасштабовану систему, так як акцент зроблено на основний функціонал.

3.1 Загальна архітектура сервісу

Схема на рисунку 3.1 зображає основні компоненти першого етапу розробки системи – до проведення робіт щодо розподілу на окремі мікросервіси та масштабування, що дозволяє сфокусуватися на втіленні основних функціональних вимог. Систему можна поділити на 2 основні частини – серверну та клієнтську.

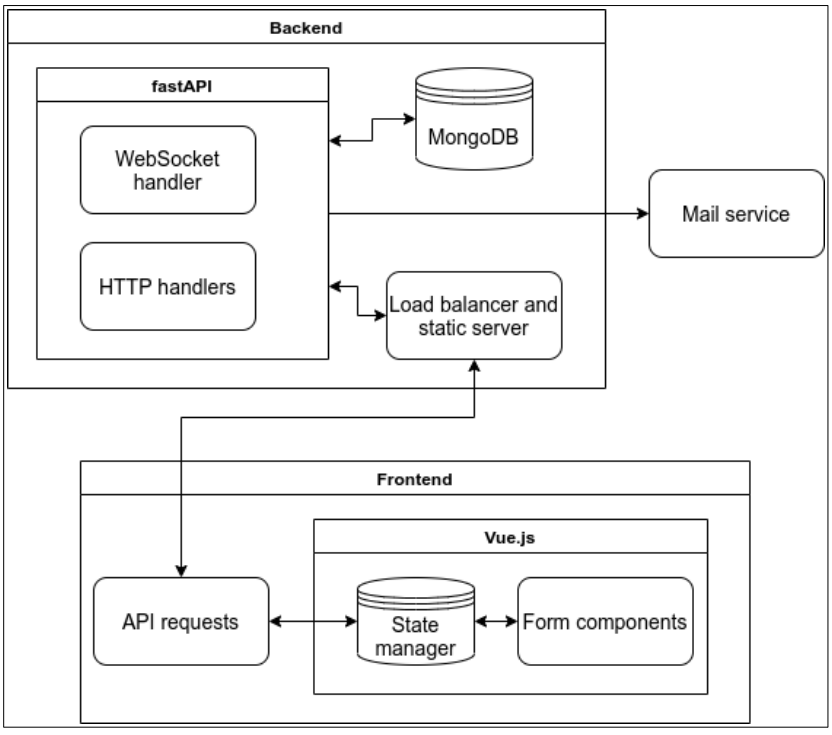


Рисунок 3.1 – Загальна архітектурна схема розроблюваного сервісу

3.1.1 Серверна частина

Серверна програмна частина складатиметься з програми сервера на базі програмної платформи fastAPI (надалі скрипт сервера, або просто сервер), що оброблятиме як загальні HTTP запити (як то реєстрація, редагування завдань, пошук проектів) так і WebSocket з'єднання для чату. Скрипт сервера буде комунікувати з базою даних MongoDB, а також зі стороннім поштовим сервером (наприклад, сервіс Mailgun). Для обробки запитів на статичні файли (зображення, файли стилів та скриптів) буде використано сервер Nginx. Задля додаткової гнучкості та централізації налаштування роботи з запитами, а також більш простого встановлення шифрування, сервер Nginx також буде використано у ролі Reverse-Proxy сервера як для HTTP, так і для WebSocket протоколу. Хоча вважаючи асинхронну природу веб-сервера Starlette, Reverse-Proxy сервер може бути зайвим – на це питання буде спроба дати відповідь у розділі про тестування.

Додатково, до етапу масштабування буде використовуватися лише 1 екземпляр обробки запитів fastAPI задля спрощення роботи зі станами чата.

3.1.2 Клієнтська частина

Основу клієнтського інтерфейсу утворюватимуть реактивні компоненти Vue, а для забезпечення обміну даними між компонентами буде використано бібліотеку керування станом Vuex, що забезпечить єдине джерело даних. У подальшому можливе використання збереження стану у localState або indexedDB (вручну, або використовуючи бібліотеку Vuex-persist [49]) , таким чином значно зменшуючи кількість запитів до сервера, проте слід враховувати ліміти виділеного дискового простору. У окремий модуль виведено набір запитів до API сервера, для яких буде використано бібліотеку Axios [47].

3.2 Схема даних БД

Так як використовується NoSQL БД об’єктного типу, що потребують максимально можливої агрегації (вкладеності) даних та дублюванням даних за необхідністю – протилежність нормалізації реляційних БД [20], можна відійти від класичних ER-схем. MongoDB може працювати у трьох режимах: строго забезпечувати дотримання схеми, видавати повідомлення (warning) у випадку неспівпадіння полей з зазначеними у схемі та повністю ігнорувати її. У будь-якому випадку, важко розроблювати систему не маючи повного уявлення про майбутню схему сутностей.

Загальний вигляд отриманої схеми з використанням Moon Modeller [7] наведено на рисунку 3.2.

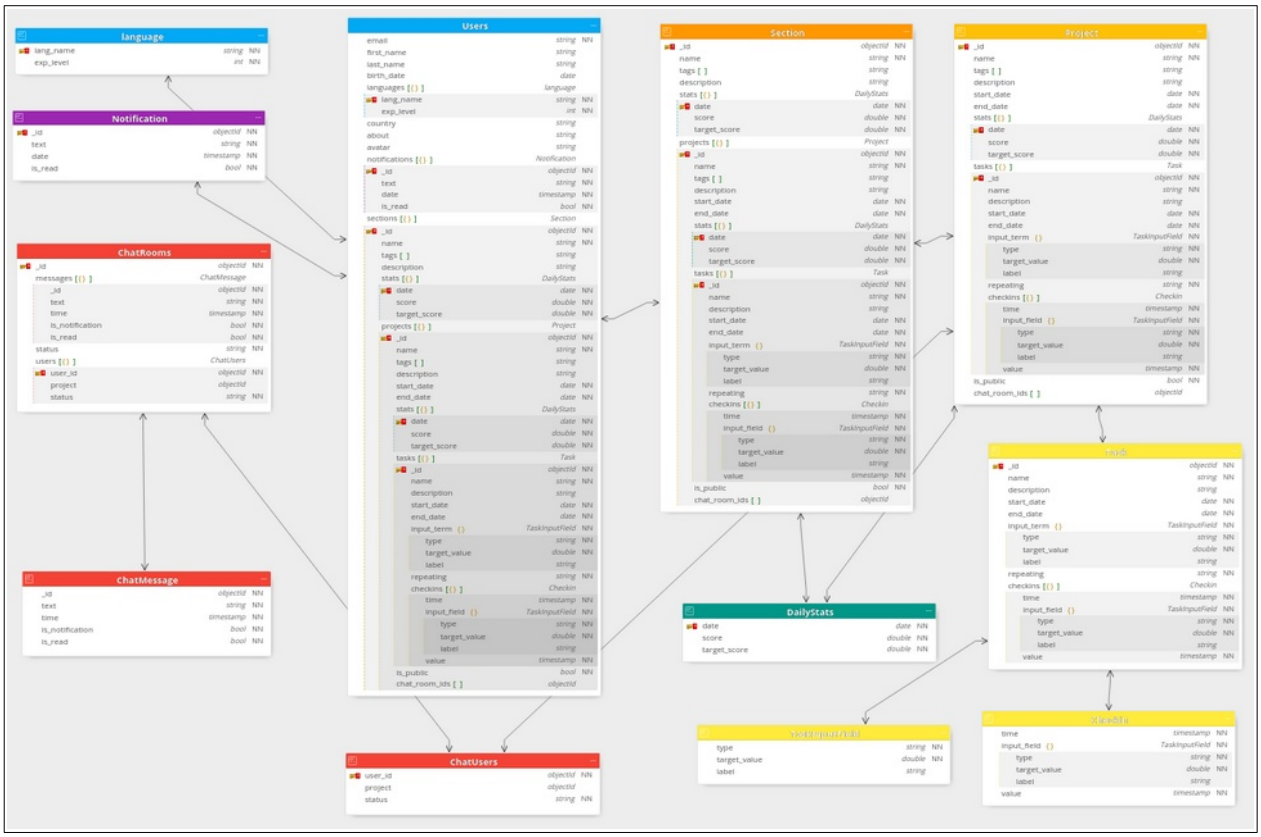


Рисунок 3.2 – Схема об’єктів БД

Рисунок 3.2 наведено для загального розуміння кількості елементів, він відображає як схеми колекції, так і окремо їх складові об’єкти. В решті отримано всього 2 колекції – Користувачі та Чати. На рисунку 3.3 зображено лише кінцеві агреговані колекції.

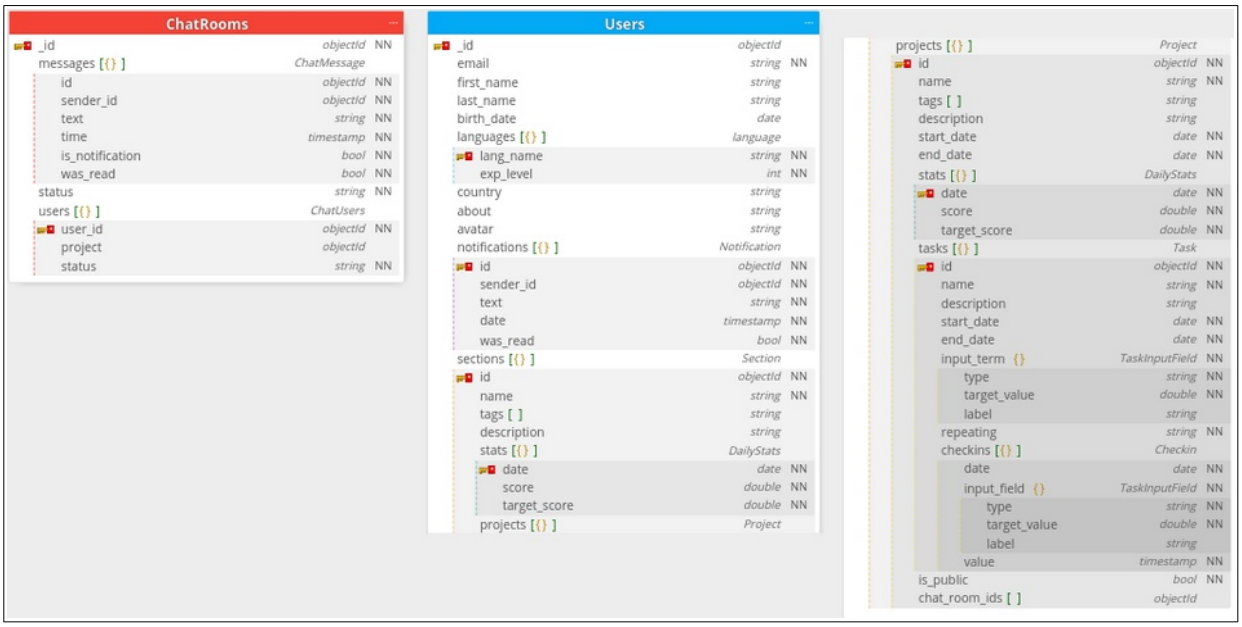


Рисунок 3.3 – Схема колекцій (остання колонка – продовження колекції Користувачів)

3.2.1 Особливості схеми колекції користувачів

Колекція користувачів має основні поля відповідно зазначених у попередньому розділі з опису концепції. На схемі відсутні деякі поля, як то поле хешованого паролю користувача та статусу активації облікового запису – вони будуть додані у процесі інтеграції бібліотеки користувачів fastAPI-users.

Поле “_id” є автоматично створюваним ідентифікатором, є необхідною складовою колекції MongoDB. Так як об’єкти (наприклад відмітки про виконання задач) є вкладеними і поєднання таблиць (Join) не потребується, то додатковий ідентифікатор “id” може виглядати зайвим, проте Vuejs для реактивного рендерингу лише змінених даних потребує поле, що може

слугувати унікальним ідентифікатором для коректної роботи у разі зміни стану відповідних даних.

Задля забезпечення коректності поля “languages.lang_name” (аналогічно з полем “languages.country”) при додаванні буде виконуватися перевірка валідної назви мови відповідно до деякого переліку допустимих значень на стороні сервера.

Значення “languages.exp_level” будуть у діапазоні від 1 до 3, що відповідатиме рівню навичок необхідних для текстової комунікації, вільного та нативного рівня володіння мови.

Поле “languages.avatar” буде містити адресу зображення на сервері, а не саме зображення, таким чином потенційно більш ефективно використовуючи кешування файлового сервера.

3.2.1.1 Сповіщення

Масив сповіщень можливо було виконати як окрему кімнату чата, але було обрано саме такий підхід, тобто вкладене поле колекції користувачів “languages.notifications”, таким чином минаючи необхідні додаткові перевірки на тип кімнати чата і трохи зменшуючи надмірність даних у вигляді поля “is_notification”, що можна буде бачити у повідомленнях кімнат чатів, але на початкових стадіях розробки з використанням об’єктних БД займатися такими незначними оптимізаціями за об’ємом не має сенсу. На жаль виникає проблема неявної залежності даних, а саме поле “was_read”: воно має відповідати однаковому значенню для сповіщення з кімнати проекту чату та кімнати агрегації сповіщень, тож будуть потрібні додаткові запити БД.

Як текст, так і спеціальні чат-об’єкти будуть зберігатися у одному полі “text”, а відокремлення цих сутностей буде виконуватися завдяки спеціальному синтаксису спеціальних чат-об’єктів, що відповідним чином буде оброблюватися компонентами користувацького інтерфейсу.

3.2.1.2 Розділи, проекти та задачі

Поля розділів, проектів та задач в цілому відповідають описаному у попередньому розділі. Проте є деякі варті уваги особливості.

Розділи та проекти кожен має свій набір статистик, а саме цільову та фактичну кількість балів по дням, що на найнижчому рівні залежить від відмічання задач у полі “sections[].projects[].tasks[].input_term”. Для агрегування цієї статистики вгору будуть використані скрипти моменту виконання після запису на стороні БД. Так як це сумарні значення за день, то поле дати і буде ключовим полем.

Відмічання копіюють поточні значення поля “...tasks[].input_term”, таким чином дозволяючи коректне відображення історії у випадку зміни цих полів у задачі. Ідентифікатором також виступає дата, таким чином дозволяючи легко змінювати значення відмітки за поточний день.

Проекти також вміщують в собі ідентифікатори відповідних кімнат чатів. Таких кімнат може буди декілька на проект, що відповідатиме зміні компаньйонів з переведенням попередньої кімнати до архівного статусу. У випадку видалення проекту відповідна кімната чату видалена не буде, якщо інший користувач все ще має відповідний зі своєї сторони проект невидаленим. У подальшому, до розділів та проектів буде додано поле статусу (активний, неактивний (у процесі створення), архівний), але наразі це поле пропущено задля спрощення розробки.

3.2.2 Особливості колекції чатів

Кімната чату матиме 3 основні поля: масив повідомлень, статус та масив користувачів.

Особливістю повідомлень є те, що поле “messages[].was_read” само по собі не вказує яким саме користувачем це повідомлення було прочитано, але враховуючи, що наразі користувачів чату може бути лише 2, це поле

відповідатиме адресату, тобто перевірка на фактичний статус прочитання буде звіряти автора повідомлення (“messages[].sender_id”) з фактичним користувачем і лише після поле “messages[].was_read”.

Незважаючи на те, що зіставлення чата та його користувачів можливе через відповідне зазначене раніше поле у проектах, кімната чату має власну копію ідентифікаторів користувачів з ідентифікаторами їх проектів та статусу їх участі. Статус може бути у чотирьох станах: активний, чат у архіві, чат у стані очікування підтвердження кооперації. Інші особливості (ситуації, коли один з користувачів видаляє проект або виходить з кооперації) було зазначено у попередньому підрозділі.

3.3 Схема точок доступу API сервера

Схема поділятиметься на схему для чату за WebSocket протоколом (набір подій), та схему для усього іншого за HTTP.

3.3.1 Схема HTTP запитів

HTTP API відповідатимуть OpenAPI специфікації, дякуючи fastAPI. Запити на отримання даних будуть виконуватися через GET метод з передачею параметрів у рядку параметрів URL тіла (наприклад /search?query=someparam), тоді як інші методи можуть використовувати тіло запита.

Слід зазначити, що на етапах розробки з виведення у експлуатацію буде встановлено шифрування, таким чином параметри GET запитів також будуть зашифровані.

Запити, що приймають id користувача розуміються для виконання запитів у тому числі й стосовно інших користувачів, так як власний id буде передаватися з кожним запитом через JWT токен.

Запити на роботу з користувачами з кореневим URL “/users” наведені у таблиці нижче.

Таблиця 3.1 – HTTP запити для “/users”

URL	Тип	Параметри тіла запиту	Опис
/{id}	GET		Запит на отримання даних користувача. Якщо запит виконується за власним ідентифікатором (або взагалі без його вказання), то буде надано доступ до додаткових приватних полів, якщо такі будуть реалізовано.
/	PATCH	{first_name, last_name, birth_date, languages...}	Запит на оновлення персональної інформації користувача. Будуть змінені лише надіслані поля.
/login	POST	{login, password}	Запит на авторизацію. Повертає JWT-токен.
/register	POST	{email, password, first_name, last_name}	Запит на реєстрацію. У разі успіху повертає пусте тіло відповіді та код 201, інакше опис помилкових полів.

Продовження таблиці 3.1

URL	Тип	Параметри тіла запиту	Опис
/confirm_registration?token={confirm_token}	POST	confirm_token	Запит на підтвердження реєстрації. Це посилання має бути отримано користувачем з його поштової скриньки.
/reset_password	POST	{email}	Запит на відновлення паролю за поштою. У будь-якому разі повертає код 202, таким чином не дозволяючи виявити існуючи email адреси (додатково, сервер виконуватиме запит у константний час: посилання відповіді та фактична обробка запиту буде виконана асинхронно).

Обробка розділів виконуватимеся відповідно за базою “/sections” (таб. 3.2). Особливість у тому, що для зміни не будуть передаватись усі данні за ієрархією “розділ->проекти->задачі->відмітки про відмічання”, якщо не вказано інакше. Це забезпечить значне скорочення трафіку.

Таблиця 3.2 – HTTP запити для “/sections”

URL	Тип	Параметри тіла запиту	Опис
/	POST	{name, description, tags}	Запит на створення нового розділу.

Продовження таблиці 3.2

URL	Тип	Параметри тіла запиту	Опис
/{id}/full	PATCH	{name, description, tags}	Запит на оновлення наданих полів (у тому числі з вкладеними сутностями) вказаного розділу
/{id}	DELETE		Запит на видалення вказаного розділу з видаленням усіх залежних сутностей (проекти та задачі, кімнати чатів)
/	GET		Запит на отримання усіх розділів.
/full	GET		Запит на отримання усіх розділів з повною структурою розділ->...->відмітки задач.

Обробка проектів знаходитиметься за базою “/projects”.

Таблиця 3.3 – HTTP запити для “/projects”

URL	Тип	Параметри тіла запиту	Опис
/?section={section_id}	POST	{name, description, tags, start_date...}	Запит на створення нового проекту за вказаним розділом.
/?section={section_id}	GET		Запит на отримання усіх проектів за вказаним розділом.
/{project_id}	PATCH		Запит на оновлення заданих полів розділу (включаючи вкладені задачі)

Продовження таблиці 3.3

URL	Тип	Параметри тіла запиту	Опис
/{project_id}/switchToSection/{section_id}	POST		Запит на зміну розділу проекту.
/{project_id}	GET		Запит на отримання проекту.
/{project_id}/full	GET		Запит на отримання проекту з вкладеними задачами.

Обробка задач знаходитиметься за базою “/tasks”.

Таблиця 3.4 – HTTP запити для “/tasks”

URL	Тип	Параметри тіла запиту	Опис
/?project={project_id}	POST	{name, description, tags, start_date...}	Запит на створення нової задачі за вказаним розділом

Продовження таблиці 3.4

URL	Тип	Параметри тіла запиту	Опис
/?project={project_id}&startDate={start_date}&end_date={end_date}	GET		Запит на отримання усіх задач за вказаним розділом та, опціонально, відфільтрованих за датою.
/{task_id}	PATCH		Запит на оновлення заданих полів задачі.
/{task_id}/checkin	POST	{date: date, value: float}	Запит на відмічання задачі.

Обробка запитів кооперації знаходитиметься за базою “/coop”.

Таблиця 3.5 – HTTP запити для “/coop”

URL	Тип	Параметри тіла запиту	Опис
/initiate?targetProj={target_project_id}&ownProj={own_project_id}	POST		Запит на ініціювання кооперації – створення нової кімнати чата та запрошення власника цільового проекту.
/searchProject?query={search_query}	GET		Запит на пошук рекомендованих для кооперації проектів. Повертає пари проектів та їх авторів.

3.3.2 Схема Websocket запитів

На відміну від HTTP протоколу, websocket не надає специфічних типів операцій (не враховуючи службових), замість цього слід визначати власні події, події на відправлення та на очікування (серверні та клієнтські). Такий функціонал реалізує бібліотека Socket.io на стороні клієнта та розширення на стороні сервера fastAPI-socketio. Концепція роботи приблизно наступна: події визначаються у тілі повідомлення за деяким полем, наприклад “{eventName: “sendMessage”, payload: {message: “Hello”}}”, і клієнт бібліотек socket.io працює саме з цими подіями.

Таблиця 3.6 – Події Websocket зі сторони клієнта

Назва події	Параметри тіла запиту	Опис
getRoomList		Запит на отримання масиву об’єктів усіх доступних кімнат.
getOnlineStatuses		Запит на отримання статусу онлайн компаньйонів з активних кімнат.
sendMessage	{chatId, text}	Запит на відправлення повідомлення.
getNottifications	{freshOnly: bool}	Запит на отримання повідомлень. Якщо freshOnly = true – отримання лише непрочитаних повідомлень.
acceptInvite	{chatId}	Запит на прийняття запрошення до чату.
declineInvite	{chatId}	Запит на відхилення запрошення до чату.
quitRoom	{chatId}	Запит на вихід з кімнати чату (розрив кооперації).

Продовження таблиці 3.6

Назва події	Параметри тіла запиту	Опис
getMessages	{chatId, freshOnly: bool, count: int, startMessageId}	Запит на отримання повідомлень. Якщо freshOnly = true – лише непрочитаних. Якщо вказано поле count, то отримання вказаної кількості останніх за часом повідомлень, інакше – усіх. Якщо вказано startMessageId, то count буде рахуватися починаючи з вказаного повідомлення, тобто повертаючи від startMessage до startMessage + count.

Події зі сторони сервера наведено у таблиці нижче.

Таблиця 3.7 – Події Websocket зі сторони сервера

Назва події	Параметри тіла запиту	Опис
newMessage	{room_id, message_id, sender_id, text, time...}	Пересилка нового повідомлення.
newInvite	{room_id}	Пересилка нового запрошення на кооперацію до відповідної кімнати.
inviteAccepted	{room_id}	Пересилка повідомлення про прийняття запиту на кооперацію у відповідній кімнаті.
inviteDeclined	{room_id}	Пересилка повідомлення про відхилення запиту на кооперацію у відповідній кімнаті.

3.4 Компоненти клієнтського інтерфейсу

Розробка Vuejs являє собою створення компонентів та їх ієрархії, налаштування маршрутизації (зіставлення URL та відповідних компонентів-відображень), а також керування станами даних цих компонентів.

3.4.1 Маршрутизація компонентів

Розробка клієнтського інтерфейсу засобами реактивної програмної платформи Vuejs починається з виділення головних та дочірніх компонентів з точки зору маршрутизації. Ефективний розподіл дозволить зменшити кількість перемалювань при змінах відображень (наприклад, при переходах по внутрішніх посиланнях), а також не завантажувати наразі не потрібні файли на стороні клієнта. На рисунку нижче зображено фрагмент коду, що визначає маршрутизацію компонентів, тобто який компонент слід відобразити у відповідність деякому URL.

```
const routes = [
    {
        path: '/welcome', component: Welcome,
        children: [
            {path: '', redirect: {name: 'login'}},
            {path: 'login', name: 'login', component: Login},
            {path: 'register', name: 'register', component: Register}]
    },
    {
        path: '', component: Main, meta: {requiresAuth: true}, redirect: {name: 'profile', params:
 {userId: 'me'}},
        children: [
            {
                path: '/profile/:userId',
                name: 'profile',
                component: Profile,
                props: route => ({id: route.params.userId || 'me'})
            },
            {path: '/logout', name: 'logout', component: Logout},
            {path: '/calendar', name: 'calendar', component: Calendar},
            {path: '/search', name: 'search', component: Search},
            {path: '/projects', name: 'projects', component: Projects},
        ]
    }
]
```

Рисунок 3.4 – Налаштування маршрутизації відображень Vuejs

Можна бачити, що у корені є 2 маршрути – “/welcome” та “/"”: вони розділяють клієнт на форму реєстрації і авторизації та на основну робочу форму з відповідними дочірніми компонентами.

Директиви “redirect” використані задля запобігання сценарію, коли користувачеві відображається головна форма, але не її дочірні елементи.

В загалі, форми утворюють наступну структуру: головна форма (App.vue) вміщує у собі директиву “<router-view/>”, що відображає дочірні компоненти (наприклад, Welcome.vue), а та у свою чергу також може мати “<router-view/>”, що буде відмальовувати дочірні вже для цього відображення компоненти.

3.4.2 Використання менеджеру станів даних Vuex

Vue використовує принцип одностороннього оновлення даних – від батьківського компоненту до дочірнього. Якщо необхідно передати дані від дочірнього компоненту, то це досягається породженням події яку має очікувати батьківський компонент. Саме по собі це може бути досить складним процесом (у випадку декількох компонентів), а значно складніше ситуація стає, коли необхідно обмінюватися даними між дочірніми компонентами з різних гілок дерева вкладеності. Для вирішення цієї проблеми використовують підхід централізованого менеджеру даних, де для Vue стандартом є додаток Vuex.

Схема взаємодії Vuex з іншими частинами застосунку наведена на рисунку нижче.

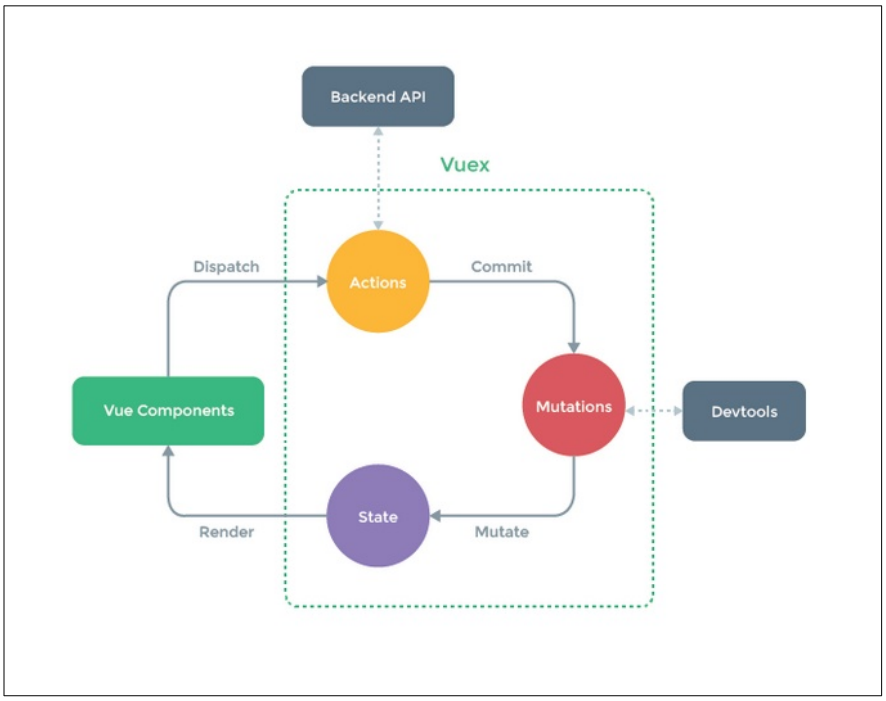


Рисунок 3.5 – Схема роботи застосунка на Vue з використанням Vuex

Серед вартих особливої уваги особливостей слід зазначити те, що мутації (методи, що напряму змінюють стан) є синхронними методами, тоді як дії (Actions) можуть буди асинхронними. Це зауваження актуальне щодо проблеми асинхронного завантаження даних із зовнішнього джерела (на схемі – Backend API), тоді як форми потребують деякий стан для відображення негайно.

Вищеописана проблема може бути вирішена з використанням наступного популярного патерну: шаблон компоненту розподілено на 2 (або 3 – для повідомлення про помилки при завантаженні) секції з використанням директиви “v-if” та “v-else” – відображення повідомлення про очікування завантаження та основної інформаційної частини. Параметром для “v-if” слугує змінна “loadingState”, яку можуть змінювати методи компонента, у даному випадку деякий з автоматично викликаємих методів для різних життєвих циклів форми – так звані хуки (hook – гак). Приклад цього можна бачити на рисунку нижче.

```
beforeMount() {
  this.$store.dispatch( type: 'loadProfile', this.$route.params.userId)
      .then(() => this.loadingState = 'done')
      .catch((err) => {
        this.loadingState = 'error';
        console.error(err)
      })
}
```

Рисунок 3.6 – Використання “гаку” beforeMount для завантаження даних профілю користувача (“/views/Profile.vue”)

Наведений на рисунку 3.6 код використовує асинхронний метод дії зі сховища (функціонал JavaScript – Promise), надаючи функцію зворотнього виклику, яка у випадку успішності операції змінює змінну стану завантаження форми на значення успіху. Завдяки реактивності, зміну цієї змінної помічає раніше описана директива “v-if="loadingState == ‘done’"” та відображає заповнену даними форму.

3.5 Висновки за розділом

У даному розділі на основі деталей опису концепту з попереднього опису було проведено розробку схеми БД, у результаті чого отримано 2 об’єктні колекції Користувачів та кімнат Чатів.

Також, було розроблено схему для HTTP та Websocket запитів, що описує API сервера для клієнтських запитів. Так як Websocket у рамках одного з’єднання не реалізує розподіл між запитами (GET, POST та інше у HTTP), було використано бібліотеку Socket.io, а необхідні запити оформлено у термінах подій.

Описано особливості розробки клієнтського коду, у тому числі питання маршрутизації та використання менеджеру станів.

РОЗДІЛ 4
МАСШТАБУВАННЯ ТА РОЗГОРТАННЯ СЕРВЕРНОЇ ЧАСТИНИ

У попередньому розділі було розглянуто деталі реалізації системи без урахування проблем масштабування задля акцентування уваги на реалізації функціональних задач з концепту сервісу. Цей розділ розглядає етап масштабування сервісу.

Розробка та налаштування масштабування сервера проведено за наступники кроками:

а) розбиття на окремі на мікросервіси;

б) контейнеризація отриманих мікросервісів;

в) проектування розподіленої архітектури для отриманих контейнейрів;

г) проектування схеми розміщення у Kubernetes:

1) налаштування Pods та Secrets;

2) встановлення MongoDB Operator та Nginx-Ingress;

3) налаштування мережевої маршрутизації між отриманими структурними елементами.

Після цього залишається етап розгортання отриманого налаштування Kubernetes у відкритих хмарних провайдерах.

Останнім кромок є налаштування домену сервіса через встановлення публічного IP сервісу у відповідні DNS записи.

4.1 Розбиття на мікросервіси

Розбиття на мікросервіси дозволяє більш ефективно балансувати навантаження на систему.

На основі схеми зображеної на рисунку 3.1 як мікросервіси було виділено наступні компоненти:

а) база даних MongoDB;

б) обробник запитів HTTP;

в) обробник WebSocket з’єднань;

г) nginx як веб-сервер для статичних файлів та Reverse-Proxy сервер.

4.1.1 Масштабування чату з використанням патерна Publisher-Subscriber

У підпункті 3.1.1 було зазначено, що задля спрощення роботи зі станами чатів, використовувався лише один потік серверу fastAPI, таким чином дозволяючи зберігати стани як свого роду глобальну змінну екземпляру сервера. Задля забезпечення розподіленої роботи двох та більше таких екземплярів стани чатів необхідно вивести у сутність вищого порядку, тобто винести у окремий сервіс. Для рішення цієї проблеми використано архітектурний патерн Publisher-Subscriber.

Ідея патерну Publisher-Subscriber наступна: деяка кількість екземплярів сервера отримують спільний доступ до компонента, що виконує роль шини даних. Цей компонент надає 2 методи – підписка на теми, та відправлення повідомлень за темами. Таким чином, у випадку чата 2-х та більше користувачів, запити яких оброблюють різні екземпляри сервера, кожний екземпляр робить підписку на повідомлення з темою id кімнати чата. Коли екземпляр отримує нове повідомлення від користувача, він надсилає повідомлення за темою чата до шини даних. Відповідно, інший екземпляр сервера, що підписаний на відповідну тему отримує повідомлення і може передати його іншому учаснику чата, якого оброблює саме цей екземпляр. Візуалізація цього процесу наведена на рисунку 4.1.

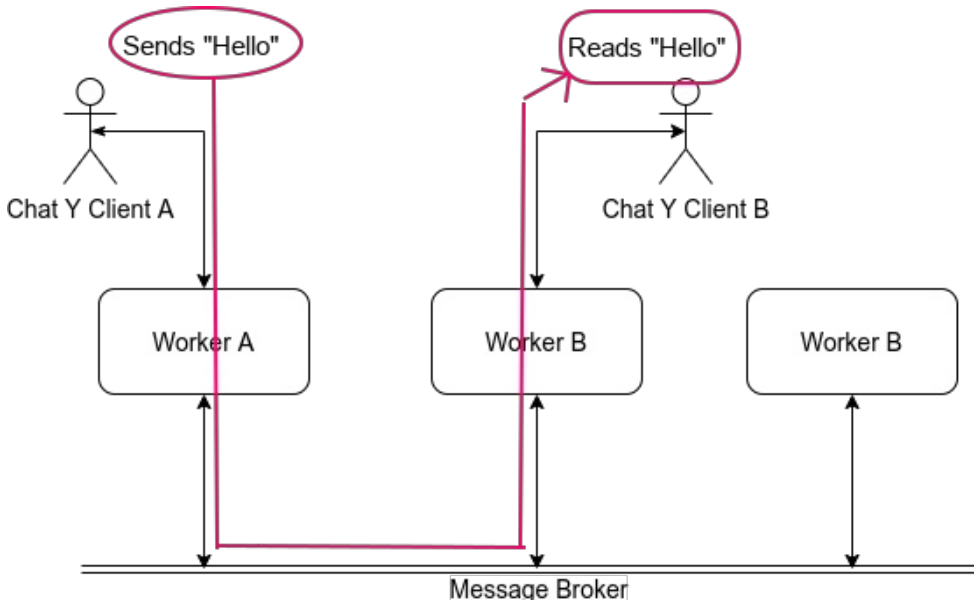


Рисунок 4.1 – Цільова пересилка повідомлення через Message Broker

У ролі такого брокеру повідомлень (Message Broker) у зв'язку з простотою використання було обрано розподілене сховище ключ-значення Redis, однією з функцій якого як раз є реалізація інтерфейсу Publisher-Subscriber. Таким чином, до переліку мікросервісів додається компонент Redis.

4.2 Схема розміщення мікросервісів у Kubernetes

На рисунку 4.2 зображено схему головних вузлів розгортки Kubernetes.

Компонент Nginx-Ingress відповідає за мережевий доступ до сервісів кластеру зі зовнішнього світу (Ingress), одночасно працюючи як балансувальник навантаження (Reverse-Proxy) для та статичний файловий сервер (Nginx). Не менш важливою функцією цього компонента є забезпечення шифрування.

MongoDB Operator – застосунок для Kubernetes що забезпечує управління кластером MongoDB – забезпечує функції Sharding та Replication, що зазначені у підпункті 1.3.4.1

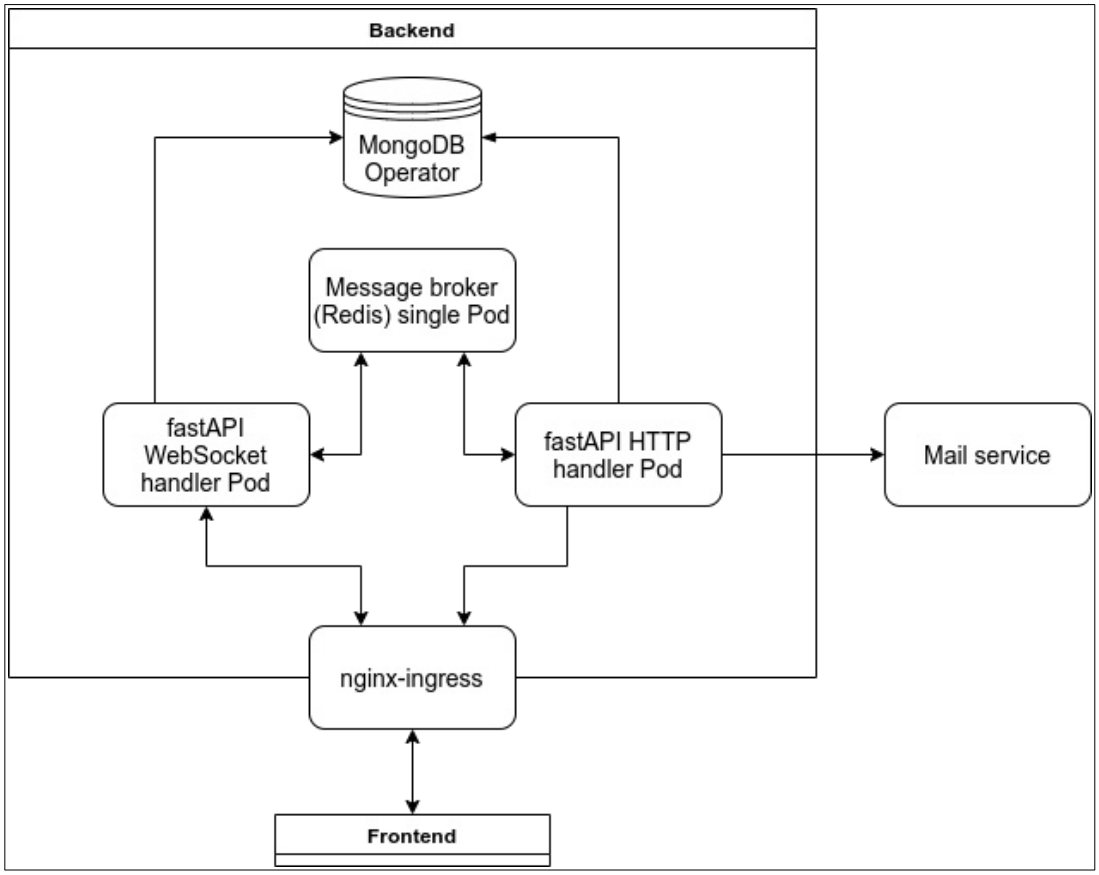


Рисунок 4.2 – Схема основних компонентів розгортки Kubernetes

fastAPI WebSocket та fastAPI HTTP фактично є одним й тим ж самим набором виконуємих файлів, різниця лише у тому, що на перший Pod Ingress направлятиме запити за протоколом WebSocket, тоді як на другий відповідно HTTP запити, таким чином декілька підвищуючи надійність системи та ефективність.

Pod для Redis представлено у одному екземплярі задля спрощення. У подальшому можливе налаштування горизонтального масштабування або заміна Redis на більш масштабуємий брокер повідомлень для сервіса чата.

Як зазначалося у пункті 3.1.1, для надсилання пошти обрано зовнішній сервіс.

ВИСНОВКИ

У даній роботі було розглянуто проблематику виконання особистих задач. Було розглянуто саме поняття задач, виділено фактори впливу на мотивацію, на основі чого проведено аналіз існуючих програмних сервісів для допомоги з особистими задачами користувачів з фокусом на соціальні фактори.

Було розглянуто основний інструментарій для розробки сучаного веб-застосунку з використанням реактивного SPA фреймворку на стороні клієнта, та засобів масштабування серверної частини.

На основі попередньо зазначеного аналізу, було виведено перелік вимог до розробки біль прогресивної системи допомоги виконання особистих задач, а саме системи групової допомоги. Проте, у подальшому виведений деталізований концепт системи дещо відрізняється від початкових вимог, що пояснюється великим об’ємом робіт, тому остаточні вимоги було зведено до виконання найбільш важливих на думку автора вимог.

Описано особливості розробки клієнтського коду з використанням Vue та менеджеру станів Vuex.

Описано підхід до масштабування системи з попереднім її розбиттям на окремі мікросервіси та схему їх розгортки на кластері Kubernetes.

Проведена робота дозволяє швидко перейти до конструювання описаного сервісу, що, на думку автора, має значний потенціал для становлення як популярного та корисного сервісу з визначення та досягнення важливих особистих цілей у новому форматі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Accountability Partners Are Great. But “Success” Partners Will Change Your Life. [Електронний ресурс]. – Режим доступу: https://medium.com/@benjaminhardy/accountability-partners-are-great-but-success-partners-will-change-your-life-8850ac0efa04
2. Amazon Web Services [Електроний ресурс].Что такое реляционная база данных – Режим доступу: https://aws.amazon.com/ru/relational-database/
3. Async IO on Linux: select, poll, and epoll [Електронний ресурс] – Режим доступу: https://jvns.ca/blog/2017/06/03/async-io-on-linux--select--poll--and-epoll/
4. Best Frontend Frameworks of 2020 for Web Development [Електронний ресурс] – Режим доступу: https://www.simform.com/best-frontend-frameworks/
5. Bootstrap documentation [Електронний ресурс] – Режим доступу: https://getbootstrap.com/docs/4.1/getting-started/introduction/
6. CAP Theorem [Електронний ресурс]. – Режим доступу: https://www.ibm.com/cloud/learn/cap-theorem
7. Data modeling tool for MongoDB, PostgreSQL, MariaDB, SQLite and GraphQL [Електронний ресурс]. – Режим доступу: https://www.datensen.com/data-modeling/moon-modeler-for-databases.html
8. Docker vs. Virtual Machines: Differences You Should Know [Електронний ресурс]. – Режим доступу: https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/
9. Document Object Model (DOM) [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

10. EKS vs GKE vs AKS – Evaluating Kubernetes in the Cloud [Електронний ресурс]. – Режим доступу: https://www.stackrox.com/post/2020/10/eks-vs-gke-vs-aks/

11. Experimenting with WebTransport [Електронний ресурс] – Режим доступу: https://web.dev/webtransport/

12. FastAPI [Електронний ресурс]. – Режим доступу: https://fastapi.tiangolo.com/

13. Filipova O. Complete Vue.js 2 Web Development / Olga Filipova., 2016. – 334 с.

14. How-To Geek [Електроний ресурс]. What Is an API – Режим доступу : https://www.howtogeek.com/343877/what-is-an-api/

15. Internet Engineering Task Force [Електронний ресурс] – Режим доступу: https://www.ietf.org/

16. JSON. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/JSON

17. Learn the differences between Shadow DOM and Virtual DOM [Електронний ресурс] – Режим доступу: https://vuejsfeed.com/blog/learn-the-differences-between-shadow-dom-and-virtual-dom

18. Long polling [Електронний ресурс]. – Режим доступу: https://javascript.info/long-polling

19. M. H. Masse REST API Design Rulebook, 2011. -116 с. – ISBN 9781449310509

20. Microsoft Docs [Електроний ресурс]. Описание основных принципов нормализации баз данных – Режим доступу: https://docs.microsoft.com/ru-ru/office/troubleshoot/access/database-normalization-description

21. Model One-to-Many Relationships with Embedded Documents [Електронний ресурс]. – Режим доступу:

https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/

22. MongoDB – Overview [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/mongodb/mongodb_overview.htm

23. Most Popular Databases [Електронний ресурс]. – Режим доступу: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/#:~:text=Most%20Popular%20Databases,-So%2C%20which%20databases&text=MySQL%20dominated%20this%20report%20with,%2C%20and%20Cassandra%20at%203.0%25.

24. New Generations at Work: Attracting, Recruiting, Retaining and Training Generation Y [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books?hl=en&lr=&id=ONdoBshH8IMC&oi=fnd&pg=PA4&dq=job+market+for+generations&ots=UfF2n639iC&sig=1ZZ-bXT1_3fkJMPUDAbKT3SxO_U&redir_esc=y#v=onepage&q=job%20market%20for%20generations&f=false

25. OpenAPI Specification [Електронний ресурс] – Режим доступу: http://spec.openapis.org/oas/v3.0.3

26. Pods [Електронний ресурс] – Режим доступу: https://kubernetes.io/docs/concepts/workloads/pods/

27. Pydantic documentation [Електронний ресурс]. – Режим доступу: https://pydantic-docs.helpmanual.io/

28. Python 3.9.1 documentation [Електронний ресурс] – Режим доступу: https://docs.python.org/3/

29. SQL vs. NoSQL Databases: What's the Difference? [Електронний ресурс]. – Режим доступу: https://www.ibm.com/cloud/blog/sql-vs-nosql

30. Self-Help Books: WHY AMERICANS KEEP READING THEM [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books?

id=lyshCWtZQgsC&dq=why+self-help+books+doesn%27t+works+study&lr=&source=gbs_navlinks_s

31. Starlette. The little ASGI framework that shines. [Електронний ресурс]. – Режим доступу: https://www.starlette.io/

32. Study Confirms Smart Strategies for Achieving Goals [Електронний ресурс]. – Режим доступу: https://sciencebeta.com/study-confirms-smart-strategies-for-achieving-goals/

33. The $10 Billion Self-Improvement Market Adjusts to a New Generation [Електронний ресурс]. – Режим доступу: https://blog.marketresearch.com/the-10-billion-self-improvement-market-adjusts-to-new-generation

34. The Gameful World: Approaches, Issues, Applications [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books?id=KDxTBgAAQBAJ&printsec=frontcover#v=onepage&q&f=false

35. The Power of SMART Goals: Using Goals to Improve Student Learning By Anne Conzemius, Jan O'Neill [Електронний ресурс]. – Режим доступу: https://books.google.com.ua/books?hl=en&lr=&id=VWwXBwAAQBAJ&oi=fnd&pg=PT15&dq=SMART+goals&ots=23HLobNwfz&sig=2c4l1txtYe3CXeVBMBP24iEVmUg&redir_esc=y#v=onepage&q=SMART%20goals&f=false

36. The WebSocket API (WebSockets) [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

37. Threads in Apache HTTP Server [Електронний ресурс] – Режим доступу: https://www.ibm.com/support/pages/threads-apache-http-server

38. Vue.js Docs [Електронний ресурс]. – Режим доступу до ресурсу: https://vuejs.org/v2/guide/

39. WebSockets vs Long Polling [Електронний ресурс]. – Режим доступу: https://www.ably.io/blog/websockets-vs-long-polling

40. Welcome to Flask – Flask DOcumentation [Електронний ресурс] – Режим доступу: https://flask.palletsprojects.com/en/1.1.x/

41. What is Kubernetes? [Електронний ресурс] – Режим доступу: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

42. What is Podman? – Podman documentation http://docs.podman.io/en/latest/

43. What is REST? [Електронний ресурс]. – Режим доступу: http://www.restapitutorial.com/lessons/whatisrest.html.

44. What is the Asynchronous Web, and How is it Revolutionary? [Електронний ресурс] – Режим доступу: https://www.theserverside.com/news/1363576/What-is-the-Asynchronous-Web-and-How-is-it-Revolutionary

45. Why Docker? [Електронний ресурс] – Режим доступу: https://www.docker.com/why-docker

46. YAML [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/YAML

47. axios. Promise based HTTP client for the browser and node.js [Електронний ресурс]. – Режим доступу: https://github.com/axios/axios

48. epoll(7) – Linux manual page [Електронний ресурс] – Режим доступу: https://man7.org/linux/man-pages/man7/epoll.7.html

49. vuex-persist [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/vuex-persist

50. Діаграма Ґанта [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D2%90%D0%B0%D0%BD%D1%82%D0%B0

51. Цель и целепологание в теории социальной организации [Електронний ресурс]. – Режим доступу: https://cyberleninka.ru/article/n/tsel-i-tselepolaganie-v-teorii-sotsialnoy-organizatsii/viewer

Додаток А

Зауваження нормконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження

ДОДАТОК Б
ПОРІВНЯННЯ СЕРВІСІВ

Таблиця Б.1 – Порівняння сервісів

Сервіс	Структуризація	Візуалізація прогресу	Грошова застава	Групова робота і соц. відповідальність	Пошук компаньйонів	Простота у використанні	Вартість
1	2	3	4	5	6	7	8
habitica.com	Поділ на звички (без прив'язки до графіка), щоденні справи і завдання з однорівневим списком підзадач.	Гейміфікація під формат рольової гри. Графіки прогресу доступні через сторонні сервіси інтеграції.	Ні. Але накопичені ігрові очки згорають при невиконанні завдань. А ще за них можна купувати внутрішньоігров і предмети.	Можливість створення групових завдань. Приєднання в гільдії для обговорення проблем.	Через список учасників гільдії. Сторінка друзів.	Відчутно перевантажений інтерфейс, що, втім, адекватно для роботи з невеликими завданнями. Складність з категоризацією завдання як звички, щоденної або завдання. Безліч додаткових налаштувань для яких потрібне звернення до довідки.	Безкоштовна для базового використання. Створення групових завдань 9 $ + 3 $ за кожного нового учасника.

Продовження таблиці Б.1

1	2	3	4	5	6	7	8
smartprogress.do	Розбиття цілі на завдання і однорівневі підзадачі із зазначенням термінів на виконання. Окреме створення разових цілей і цілей-звичок.	Завдання верхнього рівня виводяться на діаграмі Ганта. Система рівнів і досвіду. Зміни досвіду виведені в графік. Виконані завдання викреслюються за типом ToDo додатків. Календар виконання звичок. Є щоденник в контексті мети.	Можна вказувати заставу. Перевірка виконання цілей користувачами сервісу або адміністрацією, а для звичок – автоматично. Застава сплачується заздалегідь.	Можна створювати запрошення для спільних цілей. Є сторінка з опублікованими цілями. Можна коментувати чужі дненвікі. Є функціонал групового куочінга по заданих тем. Персональне переписування.	Сторінка зі списком людей з сортуванням по рейтингу і фільтрацією по імені, статтю, віком та країні. Сторінка друзів.	Інтерфейс заповнення завдань відповідає потоку заповнення. Окремі сторінки виконують свої конкретні функції. В цілому інтуїтивно зрозумілий інтерфейс.	Значна частина функціоналу обмежена безкоштовною версією в аспектах кількості і приватності. Платна підписка за 20 $ в рік. Груповий коучинг з ціною від нуля до декількох десятків доларів.

Продовження таблиці Б.1

1	2	3	4	5	6	7	8
coach.me	Робота по конкретним звичкам з звітністю по днях по типу зроблено чи ні. Пропонується вибрати з уже наявного переліку звичок за категоріями або створити нову.	Звички виведені в список на виконання на конкретний день. Є статистика щодо частоти виконання в тиждень і на місяць.	Ні.	На сторінці конкретної звички можна поставити відкрите для публіки питання. Виконання звички (з доданим повідомленням-звітом) відображається в загальному для всіх користувачів потоці виконання, що підлягає коментування та відмітками «сподобалося». На потік подій окремих користувачів можна підписуватися.	Кожна звичка має свою сторінку зі списком учасників (і питаннями). Є коучинг, пошук коучів здійснюється за категоріями аналогічно звичкам. Сторінка коучів має опис послуг, що надаються, кількість клієнтів, відгуки.	Досить простий і зрозумілий інтерфейс з упором на мобільні платформи. Коучинг відбувається у вигляді досить звичайного чата.	Робота по звичкам безкоштовна. Вартість коучингу приблизно від 25 $ в тиждень, є разові консультації за допомогою відеочату з вартістю близько сотні доларів. Підписка щотижнева.

Продовження таблиці Б.1

1	2	3	4	5	6	7	8
habitshareapp.com	Робота з конкретними звичками без подзадач. Можна вказати частоту повторення (щоденна, щотижнева, свої варіанти).	Є відображення як списку звичок з відмітками виконання за останній тиждень (к-ть виконаних днів поспіль і сумарний відсоток виконаних днів), а також у вигляді календаря. Додатково є гістограма відсотка виконань по тижнями та місяцями.	Ні.	Можна переглядати перелік звичок і їх виконання у людей зі списку друзів. В одно-два натискання можна перейти до неконтекстне чата для обговорення. На вкладці самої звички можна залишати коментарі.	Ні. Ручне додавання.	Сервіс у вигляді мобільного додатку. Чіткий поділ на секції звичок, друзів і чат. Максимально просто і ясно, без зайвих елементів.	Безкоштовно.
accountably.net	Ні.	Ні.	Ні.	Пряме спілкування в чаті з перевіряючим звітність.	Є список доступних партнерів для звітності з можливістю безкоштовно задавати питання.	Елементарний інтерфейс будується навколо чата.	Від 5 до 25 доларів в залежності від частоти звітності в тиждень. Безкоштовний пробний період в 2 тижні. Оплата щомісяця.

Продовження таблиці Б.1

1	2	3	4	5	6	7	8
beeminder.com	Задається конкретна мета з налаштуванням критерію виконання (конкретна числова мета для знаходження вище, нижче або в її деяких межах або за фактом виконання). Ручне введення даних або автоматичних збір з інших сервісів.	Графік виконання по днях з різними статистичними доповненнями, як то корелює пряма, пряма оптимального значення, пряма (або діапазон) граничних успіху значень. Статистики середнього значення за весь час, за тиждень і т.п.	Списання коштів з рахунку в разі провалу виконання завдань. Є налаштування щодо поступового збільшення ціни провалу.	Можна запросити стороннього спостерігача для підтвердження виконання завдань. Графіки виконання цілей (при включенні відповідної. опції) можуть бути відображені на загальній для всіх сторінці. В особистому профілі відображаються всі цілі і вносяться дані з коментарями до них користувача (якщо не вказано як приватно).	Способів взаємодії з іншими учасниками немає. Є форум, але він не має явної прямої прив'язки до профілю виконавця цілей.	Є безліч налаштувань, деякі з яких не очевидні і вимагають звернення до довідки. Але в цілому після початкового налаштування проблем не виникає. Багато сторонні сервіси пропоную просту процедуру настройки інтеграції їх даних.	Безкоштовної підписки у багатьох випадках буває досить, але вона обмежена максимальним числом цілей (3-5) і деякими додатковими настройками, як то смс-повідомлення, інші типи цілей (наприклад, безкоштовно не доступні цілі на зменшення цільового значення), завдання вихідних днів і т.п.

Додаток В

Вихідні коди застосунку

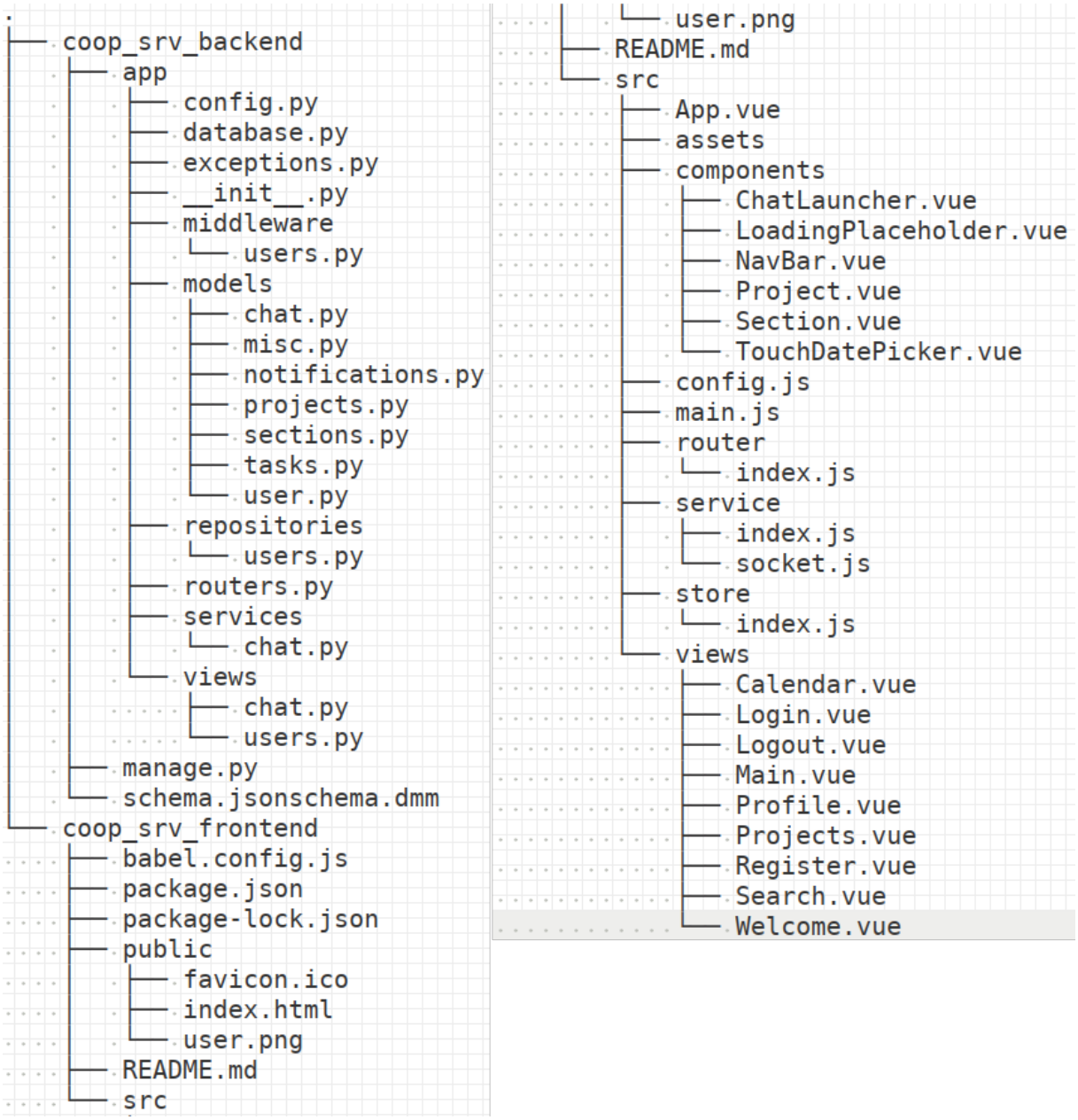


Рисунок В.1 – Структура файлів вихідних кодів

B.2 Файл: backend/app/middleware/users.py

```
from fastapi_users import FastAPIUsers
from app import database as db
from app.models import user as user_models
from fastapi_users.authentication import JWTAuthentication
from app.config import SECRET
auth_backends = []
jwt_authentication = JWTAuthentication(secret=SECRET, lifetime_seconds=3600)
auth_backends.append(jwt_authentication)
fastapi_users = FastAPIUsers(
    db.user_db,
    auth_backends,
    user_models.User,
    user_models.UserCreate,
    user_models.UserUpdate,
    user_models.UserDB,
)
```

B.3 Файл: backend/app/models/chat.py

```
from datetime import datetime
from typing import List
from uuid import UUID, uuid4
from pydantic import BaseModel, Field, UUID4
class ChatMessage(BaseModel):
    id: UUID4 = Field(default_factory=uuid4)
    sender_id: UUID
    text: str
    time: datetime = Field(default_factory=datetime.now)
class ChatUser(BaseModel):
    id: UUID4
    first_name: str
class ChatRoom(BaseModel):
    id: UUID4 = Field(default_factory=uuid4)
    name: str = "Default room"
    messages: List[ChatMessage] = []
    users: List[ChatUser] = []
```

B.4 Файл: backend/app/models/misc.py

```
from datetime import datetime
from pydantic import BaseModel
from bson.objectid import ObjectId as BsonObjectId
class DailyStats(BaseModel):
    date: datetime
    score: float
    target_score: float
class Language(BaseModel):
    lang_name: str
    exp_level: int
class PydanticObjectId(BsonObjectId):
    @classmethod
    def __get_validators__(cls):
        yield cls.validate
    @classmethod
    def validate(cls, v):
        if not isinstance(v, BsonObjectId):
            print(v)
            raise TypeError('ObjectId required')
        return str(v)
    def __str__(self):
        return str(self)
    @classmethod
    def __modify_schema__(cls, field_schema):
        print(field_schema)
```

```
        # __modify_schema__ should mutate the dict it receives in place,
        # the returned value will be ignored
        field_schema.update(
            format="uuid4",
            type="string",
            description="BsonObjectId generated by MongoDB"
        )
```

B.5 Файл: backend/app/models/notifications.py

```
from datetime import datetime
from pydantic import BaseModel
from app.models.misc import PydanticObjectId
class Notification(BaseModel):
    id: PydanticObjectId
    sender_id: PydanticObjectId
    text: str
    timestamp: datetime
    was_read: bool
```

B.6 Файл: backend/app/models/projects.py

```
from datetime import datetime
from typing import List, Optional

from pydantic import BaseModel

from app.models.misc import DailyStats, PydanticObjectId
from app.models.tasks import Task


class Project(BaseModel):
    id: PydanticObjectId
    name: str
    tags: Optional[List[str]] = []
    description: str
    is_public: bool
    start_date: datetime
    end_date: datetime
    chat_room_ids: Optional[List[PydanticObjectId]] = []
    stats: Optional[List[DailyStats]] = []
    tasks: Optional[List[Task]] = []
```

B.7 Файл: backend/app/models/sections.py

```
from typing import List, Optional
from pydantic import BaseModel, Field, UUID4
from bson.objectid import ObjectId
from uuid import UUID, uuid4
from app.models.misc import DailyStats, PydanticObjectId
from app.models.projects import Project
class Section(BaseModel):
    id: UUID4 = Field(default_factory=uuid4)
    name: str
    tags: Optional[List[str]] = []
    description: str
    stats: Optional[List[DailyStats]] = []
    projects: Optional[List[Project]] = []
```

B.8 Файл: backend/app/models/tasks.py

```
from datetime import datetime
from typing import List, Optional, Type
from pydantic import BaseModel
```

```
from bson.objectid import ObjectId
from app.models.misc import PydanticObjectId
class TaskInputTerm(BaseModel):
    type: str
    target_value: float
    label: str
class Checkin(BaseModel):
    date: datetime
    input_term: TaskInputTerm
    value: float
    target_value: float
class Task(BaseModel):
    id: PydanticObjectId
    name: str
    tags: Optional[List[str]] = []
    description: str
    start_date: datetime
    end_date: datetime
    input_term: TaskInputTerm
    repeating: str
    checkins: Optional[List[Checkin]] = []
```

B.9 Файл: backend/app/models/user.py

```
from pathlib import Path
from typing import List, Optional
from datetime import datetime
from fastapi_users import models
from pydantic import BaseModel, UUID4, validator
from app.config import DEFAULT_AVATAR_PATH
from app.models.misc import Language
from app.models.notifications import Notification
from app.models.sections import Section
class User(models.BaseUser):
    is_active: bool = True
    first_name: str
    last_name: str
    birth_date: Optional[datetime]
    languages: Optional[Language]
    country: Optional[str]
    about: Optional[str]
    avatar: str = DEFAULT_AVATAR_PATH
    notifications: Optional[List[Notification]] = []
    sections: Optional[List[Section]] = []
class UserPublic(BaseModel):
    id: UUID4
    first_name: str
    last_name: str
    birth_date: Optional[datetime]
    languages: Optional[Language]
    country: Optional[str]
    about: Optional[str]
    avatar: str = DEFAULT_AVATAR_PATH
class UserCreate(models.BaseUserCreate):
    first_name: str
    last_name: str
    @validator('password')
    def valid_password(cls, v: str):
        if len(v) < 6:
            raise ValueError('Password should be at least 6 characters')
        return v
```

```
class UserUpdate(User, models.BaseUserUpdate):
    pass
class UserDB(User, models.BaseUserDB):
    pass
```

B.10 Файл: backend/app/services/chat.py

```
from app.database import chat_coll
from app.models.chat import ChatRoom, ChatUser, ChatMessage
async def get_messages():
    return (await chat_coll.find_one({}))['messages']
async def init_main_chat():
    if await chat_coll.count_documents({}) == 0:
        await chat_coll.insert_one(ChatRoom().dict())
async def append_user(user):
    chat: ChatRoom = await chat_coll.find_one({})
    await chat.users.append(ChatUser(user))
async def append_message(user, message):
    chat: ChatRoom = await chat_coll.find_one({})
    message = ChatMessage(sender_id=user.id, text=message)
    await chat_coll.update_one(chat, {"$push": {"messages": message}})
```

B.11 Файл: backend/app/views/chat.py

```
import socketio
from app import CORS_ACCEPT_ORIGIN
from app.database import user_db
from app.middleware.users import jwt_authentication
from app.services import chat as chat_srvc
sio = socketio.AsyncServer(async_mode="asgi",
cors_allowed_origins=CORS_ACCEPT_ORIGIN)
sio_app = socketio.ASGIApp(socketio_server=sio, socketio_path='socket.io')
USERS = {}
def check_auth(func):
    async def inner(sid, *args, **kwargs):
        if sid not in USERS:
            print(f'check auth failure for sid {sid}')
            return await sio.emit('authentication', 'error')
        func(sid, *args, **kwargs)
    return inner
@sio.on('send_message')
async def send_message(sid, message):
    chat_srvc.append_message(USERS[sid], message)
@sio.on('disconnect')
def disconnect(sid, *args):
    try:
        del USERS[sid]
    except KeyError:
        pass
    print(sid, 'disconnected')
@sio.event
async def authenticate(sid, jwt: str):
    user = await jwt_authentication(jwt, user_db)
    if not user:
        print('authentication error – no user', sid)
        return await sio.emit('authentication', 'error')
    USERS[sid] = user
    await sio.emit('authentication', 'success')
@sio.on('retrieveMessages')
async def retrieve_messages(sid):
    print('retrieve_messages emitted')
    await sio.emit('old_messages', {"msg": await chat_srvc.get_messages()})
```

```
    print('old_messages emitted')
```

B.12 Файл: backend/app/views/users.py

```
from fastapi import APIRouter, Depends, HTTPException, status
from pydantic import UUID4
from app.middleware.users import fastapi_users
from app.models.user import UserPublic
from app.database import user_db
users_router = APIRouter()
@users_router.get('/{user_id}/public', response_model=UserPublic,
                  dependencies=[Depends(fastapi_users.get_current_user)])
async def get_user(user_id: UUID4):
    user = await user_db.get(user_id)
    if user is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND)
    return user
```

B.13 Файл: backend/app/__init__.py

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.config import CORS_ACCEPT_ORIGIN
from app.middleware.users import fastapi_users, jwt_authentication
from app.views.users import users_router
from app.views.chat import sio_app
from app.services.chat import init_main_chat
app = FastAPI()
app.mount('/chat', sio_app)
@app.on_event("startup")
async def startup_event():
    await init_main_chat()
app.add_middleware(
    CORSMiddleware,
    allow_origins=CORS_ACCEPT_ORIGIN,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
app.include_router(
    fastapi_users.get_auth_router(jwt_authentication),
    prefix="/auth/jwt",
    tags=["auth"],
)
app.include_router(
    fastapi_users.get_register_router(),
    prefix="/auth",
    tags=["auth"],
)
app.include_router(
    fastapi_users.get_users_router(),
    prefix="/users",
    tags=["users"],
)
app.include_router(users_router,
                   prefix="/users",
                   tags=["users"])
```

B.14 Файл: backend/app/config.py

```
DEFAULT_AVATAR_PATH = '/user.png'
DATABASE_URL = "mongodb://localhost:27017"
SECRET = "SECRET"
```

```
CORS_ACCEPT_ORIGIN = '*'
```

B.15 Файл: backend/app/database.py

```
from motor import motor_asyncio
from motor.core import AgnosticCollection
from fastapi_users.db import MongoDBUserDatabase
from app.models.user import UserDB
from app.config import DATABASE_URL
client = motor_asyncio.AsyncIOMotorClient(DATABASE_URL,
uuidRepresentation="standard")
db = client["accountability"]
user_coll: AgnosticCollection = db["Users"]
chat_coll: AgnosticCollection = db["ChatRooms"]
user_db = MongoDBUserDatabase(UserDB, user_coll)
```

Додаток Г

Презентаційний матеріал

Г.1 Слайд 1

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Науково-навчальний інститут комп'ютерних наук та технологій
Кафедра прикладної математики та інформатики

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

«Балансування навантаження в розподіленій системі групової взаємопідтримки реалізації особистих завдань»

Виконав:
студент групи ІПЗм-19
Омельченко І. А.

Науковий керівник:
д.т.н., проф. Дмитрієва О. А.

Г.2 Слайд 2

ОБ'ЄКТ, ПРЕДМЕТ, МЕТА

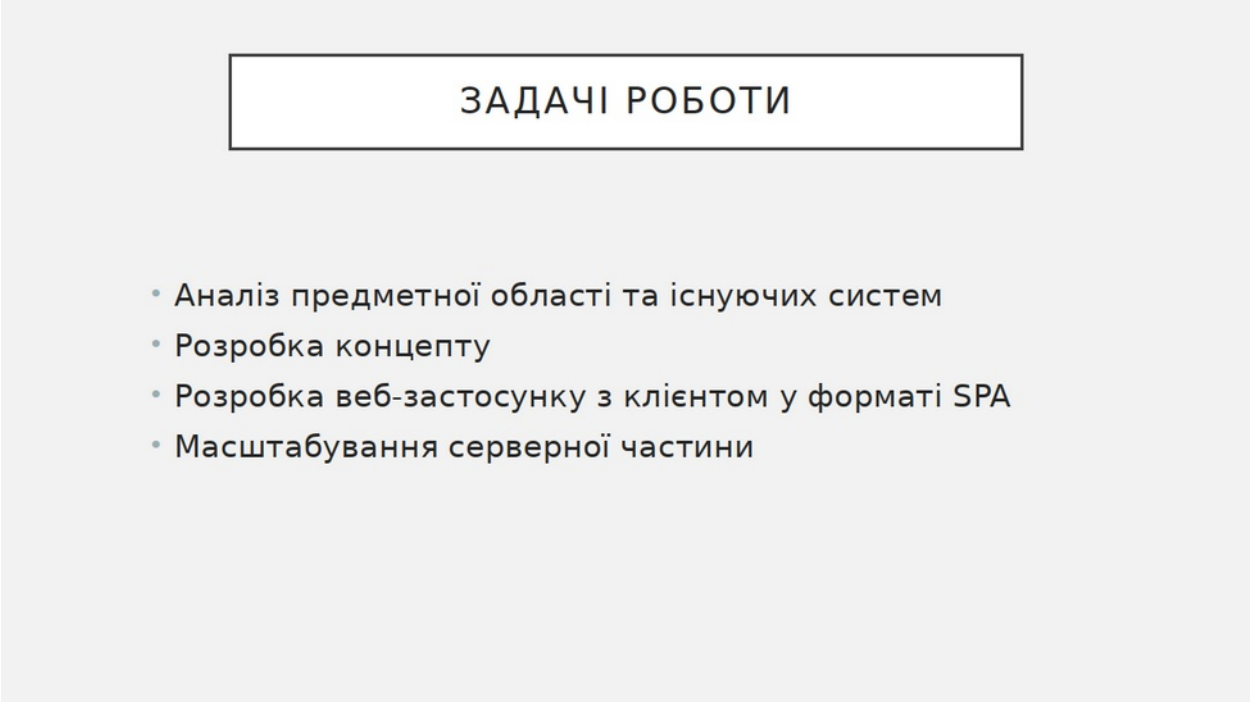
Об'єкт: процеси балансування навантаження в розподіленій системі організації особистих поточних справ користувачів.

Предмет: програмні інструменти підтримки мотивації при досягненні окремих цілей користувачів з використанням групової взаємопідтримки.

Мета: розробка розподіленої системі сервісу групової взаємопідтримки виконання особистих завдань у форматі веб-застосунку з використанням технологій горизонтального масштабування навантаження зі сторони сервера, та розробки клієнтської частини у форматі SPA.

Г.3 Слайд 3

Г.4 Слайд 4



Г.5 Слайд 5

КОНЦЕПТ РОЗРОБЛЮВАНОГО СЕРВІСУ

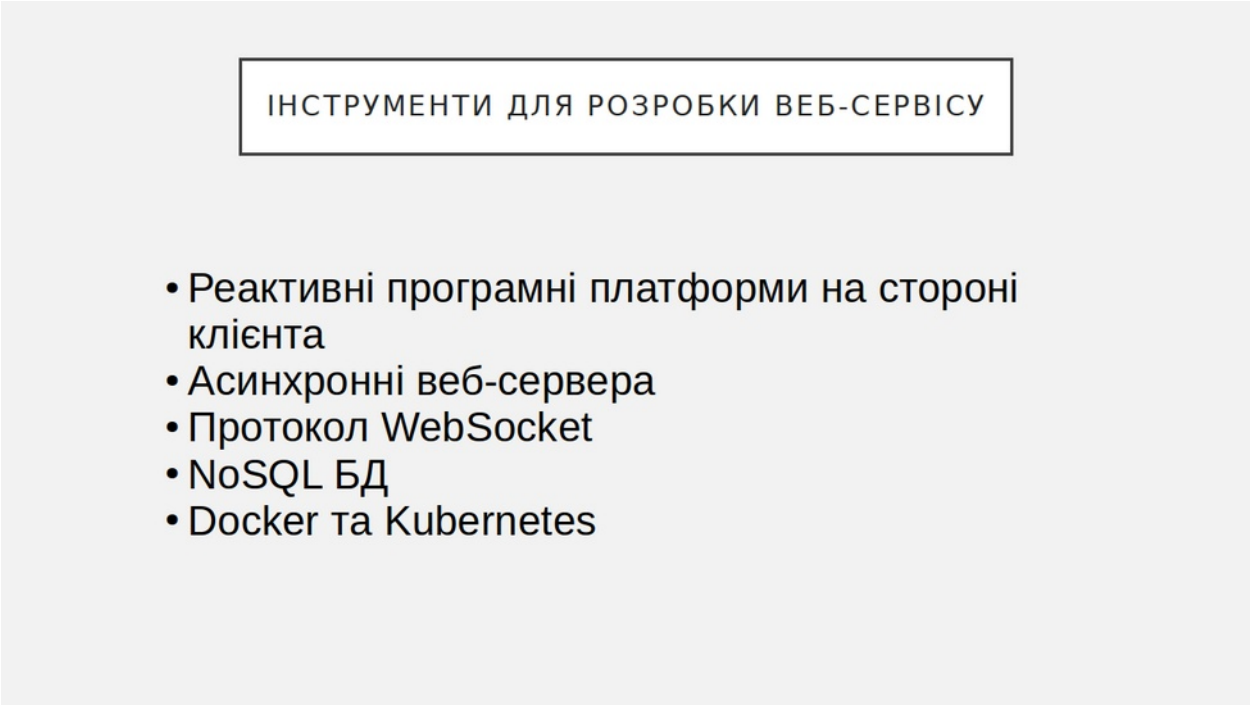
- реєстрація та авторизація;
- особистий профіль користувача;
- особистий перелік розділів, проектів та задач;
- пошук проектів інших користувачів для кооперації;
- чат;
- система повідомлень;
- система автоматичного підбору користувачів;
- календар задач.

Г.5 Слайд 5

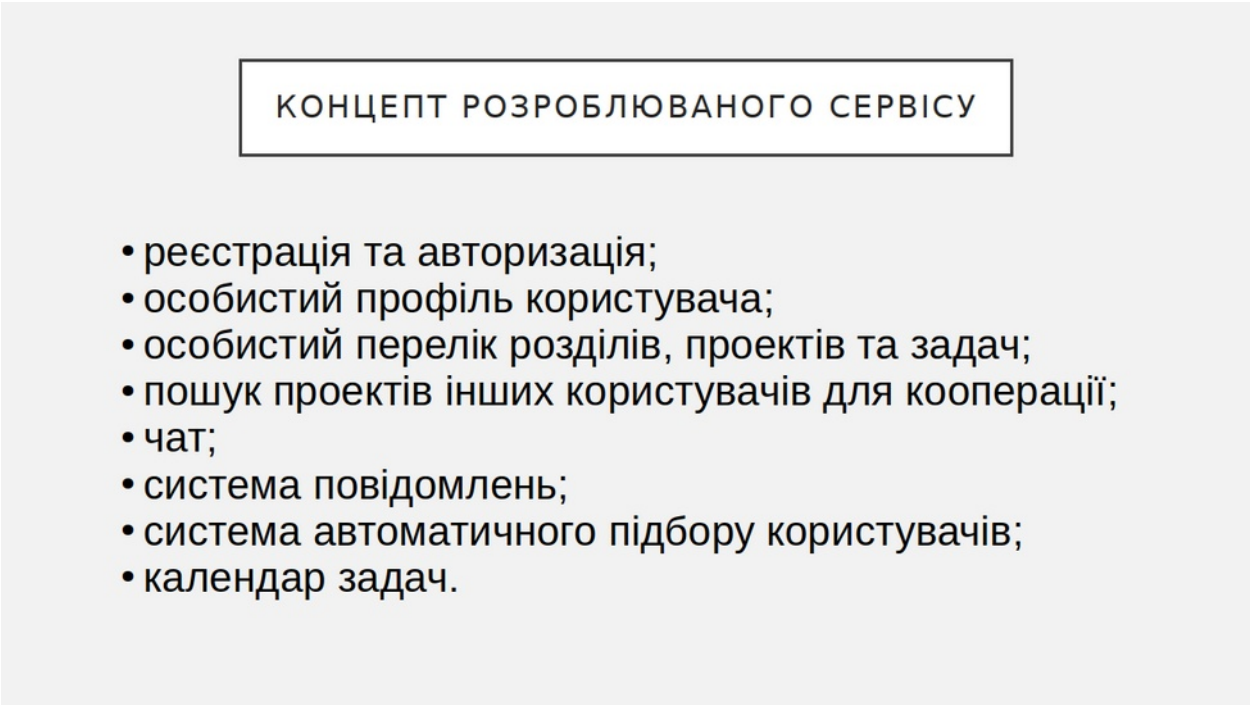
ФАКТОРИ ВПЛИВУ НА МОТИВАЦІЮ

- Структуризація планів та атомарні задачі
- Візуалізація прогресу
- Фактори негативної мотивації
- Фактор публічної звітності

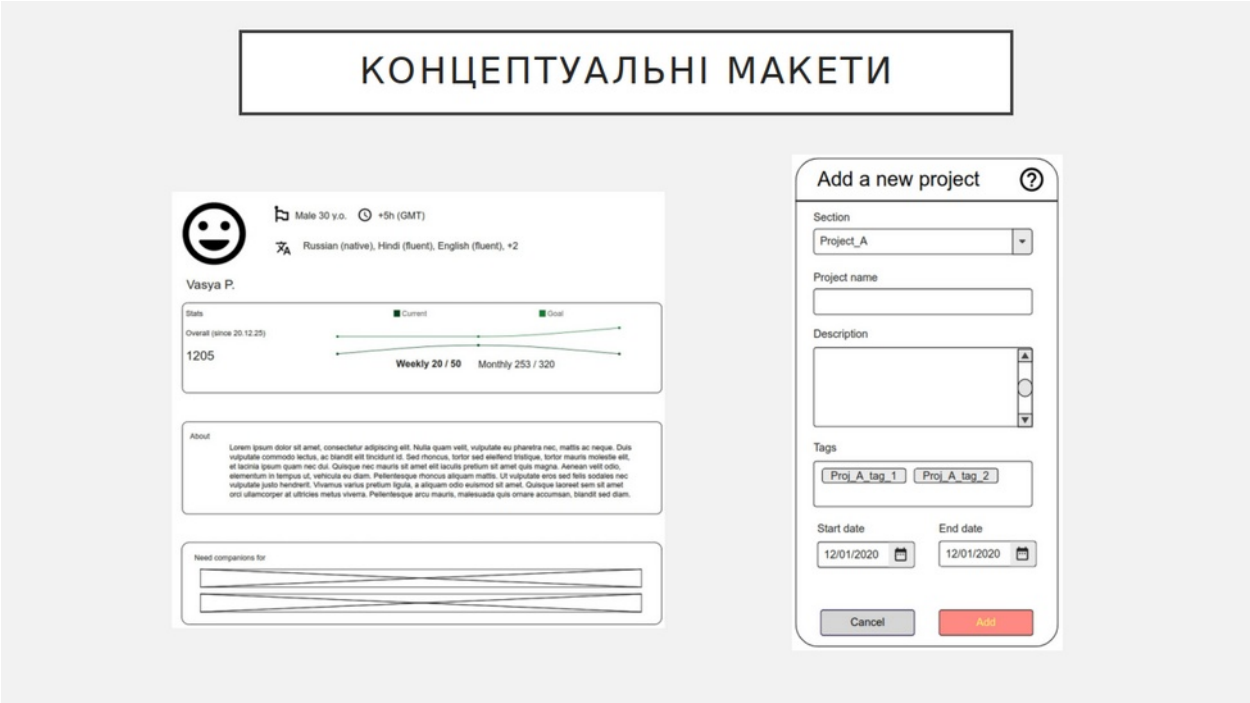
Г.6 Слайд 6



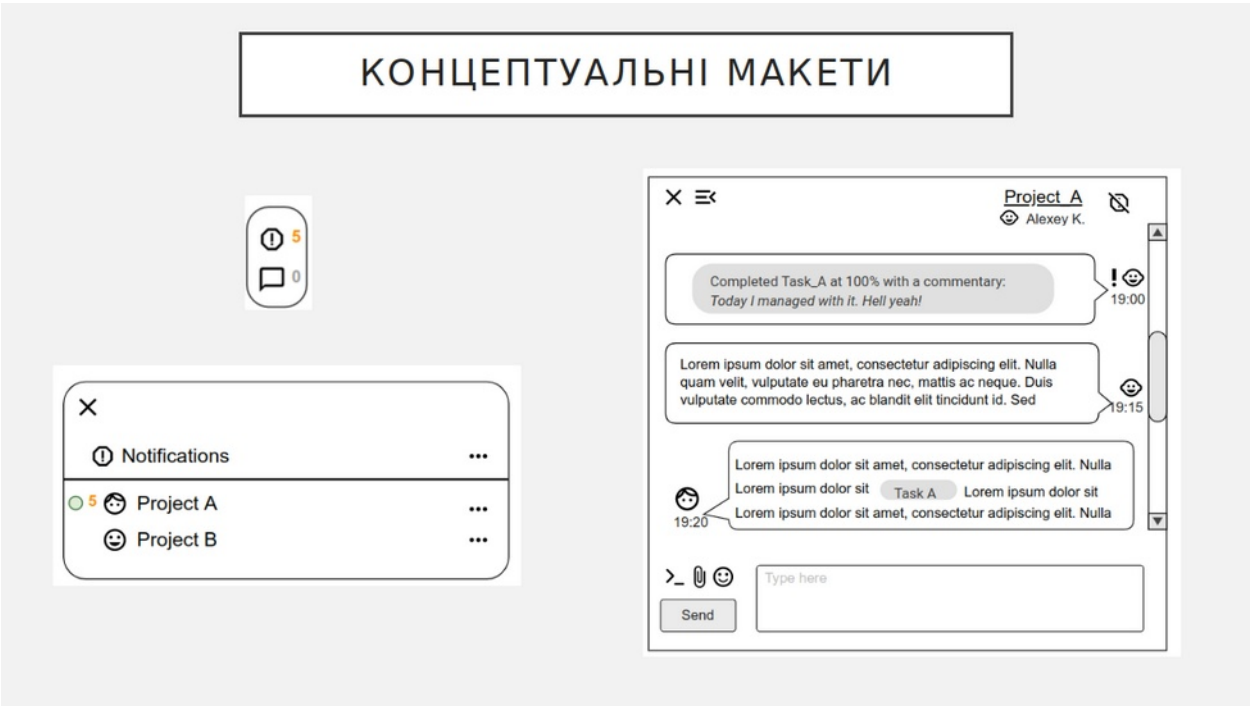
Г.7 Слайд 7



Г.8 Слайд 8



Г.9 Слайд 9

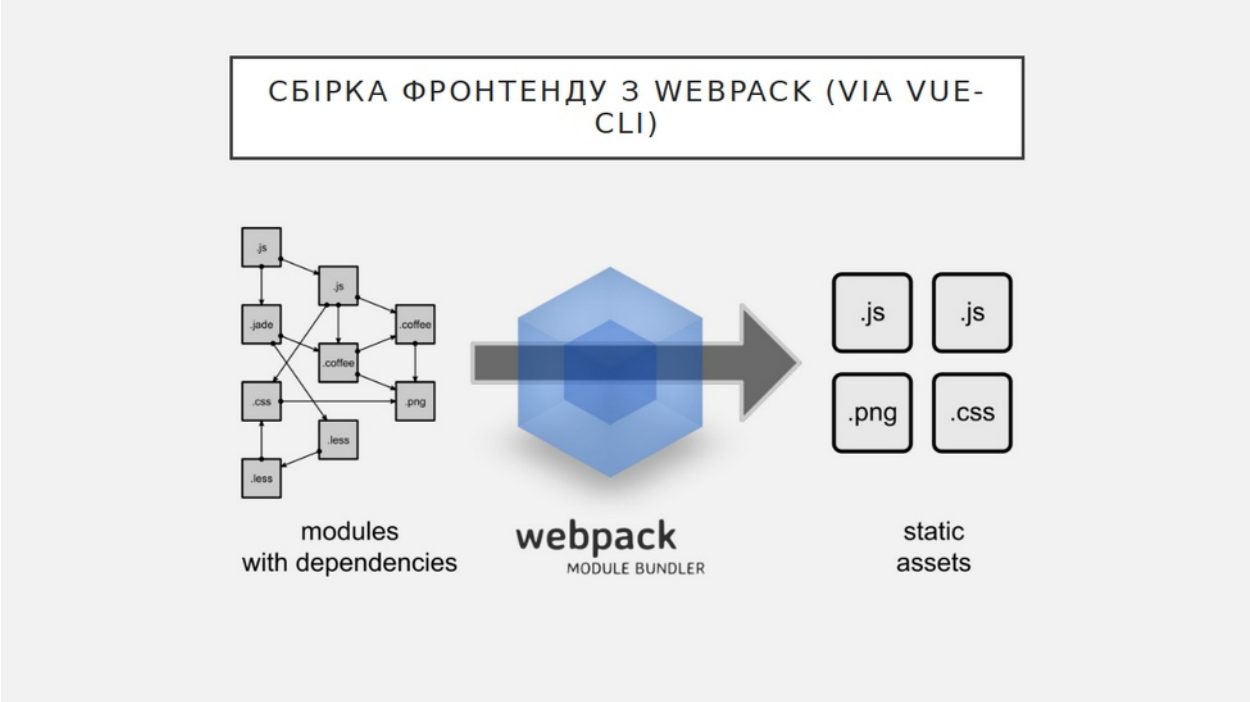


Г.10 Слайд 10

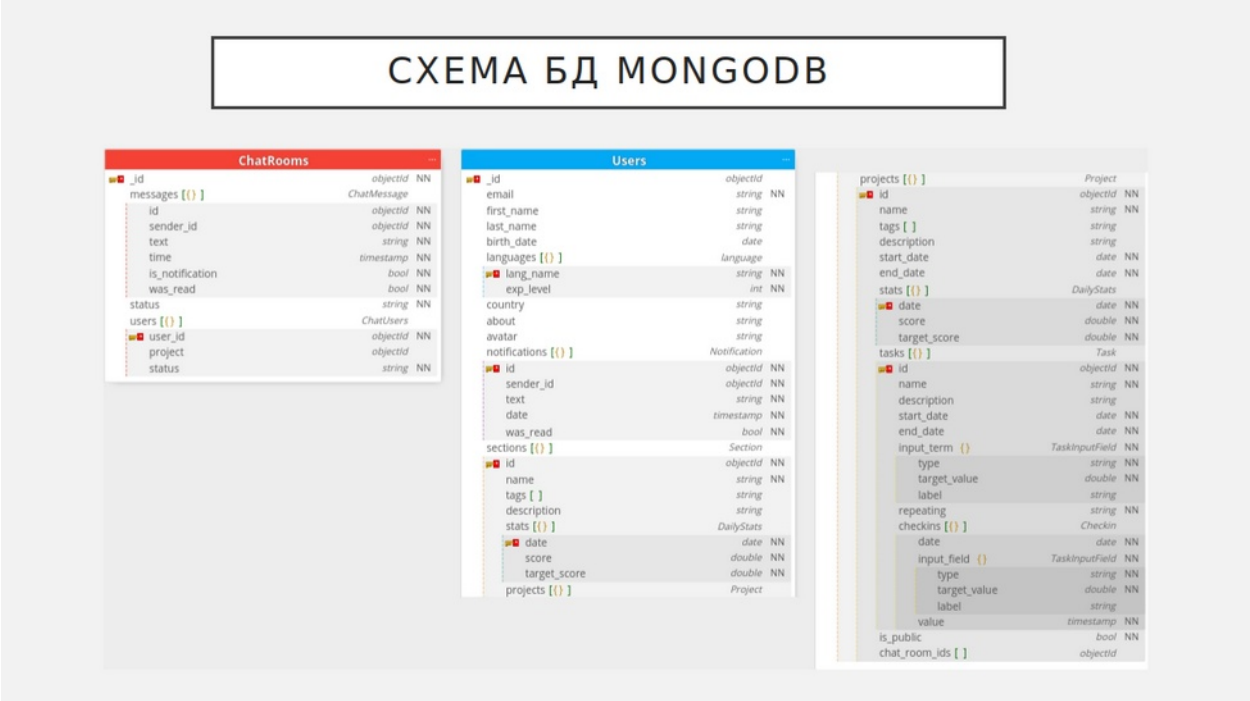
Г.11 Слайд 11

Г.12 Слайд 12

Г.13 Слайд 13



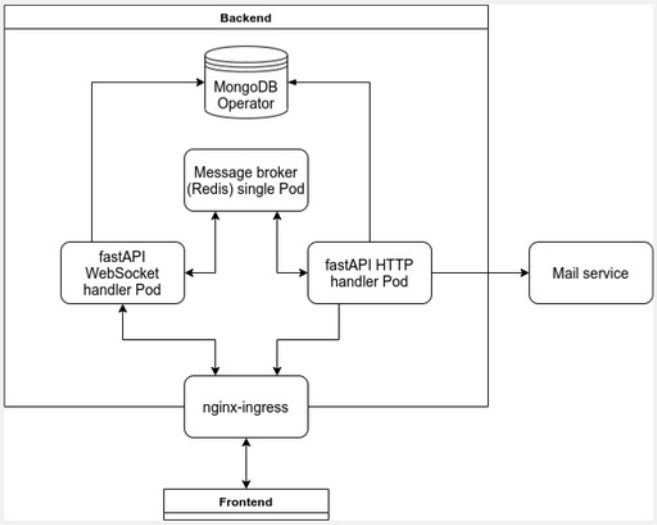
Г.14 Слайд 14



Г.15 Слайд 15

Г.16 Слайд 16

РОЗГОРТКА KUBERNETES



Г.17 Слайд 17

ВИСНОВКИ

- Виконано аналіз проблеми реалізації особистих завдань
- Створено концепт сервісу
- Реалізовано основний функціонал сервісу
- Реалізовано розподілення навантаження

Г.18 Слайд 18