Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

РОЗРОБКА ВЕБ-ТРЕКЕРУ КОРИСНИХ ЗВИЧОК

Текстова частина до курсової роботи
за спеціальністю 122, «Комп'ютерні науки»

Керівник курсової роботи
Калітовський Б.В.

Виконала студентка 3 курсу
Купчик А.В.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Кафедра мережних технологій

ЗАТВЕРДЖУЮ
Зав. Кафедри
Малашонок Геннадій Іванович
_____
*(підпис)*_____
“____” _____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентки Купчик Аліни Віталіївни факультету інформатики 3 курсу

Тема: Розробка веб-трекеру корисних звичок

Зміст ТЧ до курсової роботи:
Календарний план
Вступ
Розділ 1: Аналіз предметної області. Постановка завдання курсової роботи
Розділ 2: Теоретичні відомості
Розділ 3: Опис реалізації програмного продукту
Висновки
Список використаної літератури
Додатки

Дата видачі “____” _____ 2020 р.
Керівник_____ Завдання отримала _____

(підпис) *(підпис)*

Календарний план виконання курсової роботи

Тема: Розробка веб-трекеру корисних звичок

Календарний план виконання роботи:

№ п/п	Назва етапу курсового проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	Листопад 2020	
2.	Ознайомлення з існуючою інформацією по темі	Грудень 2020	
3.	Ознайомлення з існуючими системами аналогами	Січень 2021	
4.	Початок створення практичної частини	Лютий 2021	
5.	Початок написання теоретичної частини	Березень 2021	
6.	Аналіз практичної частини; її корегування	Квітень 2021	
7.	Подання проміжної версії практичної частини	Травень 2021	
8.	Остаточне завершення написання теоретичної частини роботи та розробки практичної частини; корегування	Травень 2021	
9.	Створення презентації	Травень 2021	
10.	Захист курсової роботи	Після 24 травня	

Студентка Купчик А.В.

Керівник Калітовський Б.В.

“_____” _____

ЗМІСТ

Перелік умовних позначень

BeeNow - назва застосунку, яке розроблялось під час написання курсової роботи

Челендж - виклик/випробування, яке створюють користувачі у даному веб-застосуванні.

Справа - одноразова дія, яку можна виконати в додатку в розділі To-do.

Юзер - те саме, що і користувач

Дедлайн - те саме, що і кінцевий термін здачі

ВСТУП

Не буде перебільшенням стверджувати, що питання планування власного часу та набуття корисних звичок на даний момент є актуальним серед більшості людей. Через постійний хаос та нескінченний перелік справ, ми забуваємо правильно харчуватись, займатись спортом, прочитати розділ улюбленої книги тощо. Все відкладається на завтра, яке ніколи не настає. Безперечно, у кожного з нас є бажання, до реалізації яких ми ніколи не доходимо, наприклад, приділити час вивченню нової мови, або ж просто вийти прогулятись і насолодитись хорошою погодою. Завжди є щось, що заважає розпочати. Коренем такої проблеми є неправильний розподіл пріоритетів та погана самодисципліна. Одним з можливих рішень даного питання є веб-застосунок “BeeNow”.

Метою курсової роботи є створення автоматизованої системи трекінгу звичок, яка допоможе візуалізувати справи, розвинути власну самодисципліну та працелюбність.

Основним завданням проекту є детальне дослідження предметної області, на основі якого відбудеться розробка веб-застосування, інструментарій якого буде включати в себе найважливіші функції для полегшення вироблення нових звичок у користувачів.

Об’єктом дослідження є існуючі веб та мобільні додатки, які спрямовані на розвиток корисних звичок та позбавлення від шкідливих. На основі аналізу функціоналу додатків-аналогів буде розроблено інструментарій веб-застосування “BeeNow”.

Робота складається з наступних частин: вступ, 3 основні розділи, висновки, список використаної літератури та додатки.

У першому розділі здійснено аналіз сучасного стану питання, обґрунтування обраної теми, пошук та дослідження додатків-аналогів та формулювання постановки завдання.

У другому розділі здійснено аналіз теоретичних відомостей відносно створення веб-застосувань. Наведено приклад та обґрунтування використаної

архітектури, що використовувалась під час проектування структури додатку “BeeNow”.

У третьому розділі здійснено детальний опис реалізації програмного продукту, включаючи аналіз технічного завдання, у якому описано специфікацію вимог до даних, опис та обґрунтування використаних засобів розробки, опис з детальним планом дій реалізації додатку та бази даних. Крім того, у пункті “Інструкція користувача” продемонстровано наявні можливості користувача у створеному застосуванні.

РОЗДІЛ 1

1.1 Аналіз сучасного стану питання та обґрунтування теми

На сьогодні існує теорія, що для формування і закріплення нової звички необхідно витратити 21 день. Ця гіпотеза була науково доведена в США за допомогою спеціального експерименту. Його суть полягала в наступному: група людей носила окуляри з вбудованими лінзами, які перевертали світ з ніг на голову. Спочатку експеримент планувався на місяць, причому люди повинні були носити ці окуляри цілодобово.За результатами дослідження, саме 21-го дня було достатньо людському мозку для того, щоб сприйняти таке зображення як реальне. Ось звідки з’явилось припущення, що аналогічне правило працює щодо вироблення нової звички. Проте, наразі не все так просто та однозначно. Наука досі не знайшла чіткої відповіді на питання, скільки ж часу потрібно для формування звички. Тому ми не можемо стверджувати, що «правило 21 дня» працює завжди і для всіх, все індивідуально і залежить від складності самої навички. Дія стає звичкою тоді, коли виконується без протидії з боку організму. Тому різні дії потребують різного часу формування.

Для того, аби відслідковувати свій прогрес у виконанні певної дії існує так званий трекер звичок, який є зручним інструментом для впровадження нового корисного ритуалу у життя. Його основні завдання - тренувати навички самодисципліни, навчитись розставляти правильно власні пріоритети і тим самим розподілити свій час правильно та з користю. Є два варіанти ведення трекера звичок — паперовий(заповнення таблиць власноруч) та електронний(використання спеціального додатку). Перший варіант підходить більш тим людям, які отримують задоволення від створення чогось нового своїми руками та готові приділяти час на виготовлення власного трекеру, адже спираючись на свій власний досвід, це досить творчий та часоємний процес. У іншому випадку варто використовувати другий варіант та користуватись готовими додатками.

На мою думку, дана тема є досить актуальною для усіх людей, які хоча б раз у своєму житті зіштовхувались із проблемою, пов’язаною з неструктурованим підходом до завдань, невмінням правильно розподіляти свій

час, та, як наслідок, не встигати виконувати завдання вчасно і залишати власні плани не реалізованими. Я вважаю, що рішенням даної проблеми є чітке планування та обмежені часові рамки. Даний веб-застосунок допоможе користувачу визначити свою ціль, поставити для неї дедлайн та розпочати роботу.

1.2 Огляд існуючих аналогів розробки

Провівши пошук по мережі, було виявлено велику кількість різноманітних додатків-аналогів, як мобільних, так і веб-застосунків. Кожен із них мав певний спільний функціонал з іншими додатками, проте й у кожному була своя особливість. Розглянемо наступні аналоги: Habitify, Coach.me, TickTick, Strides.

1.Habitify

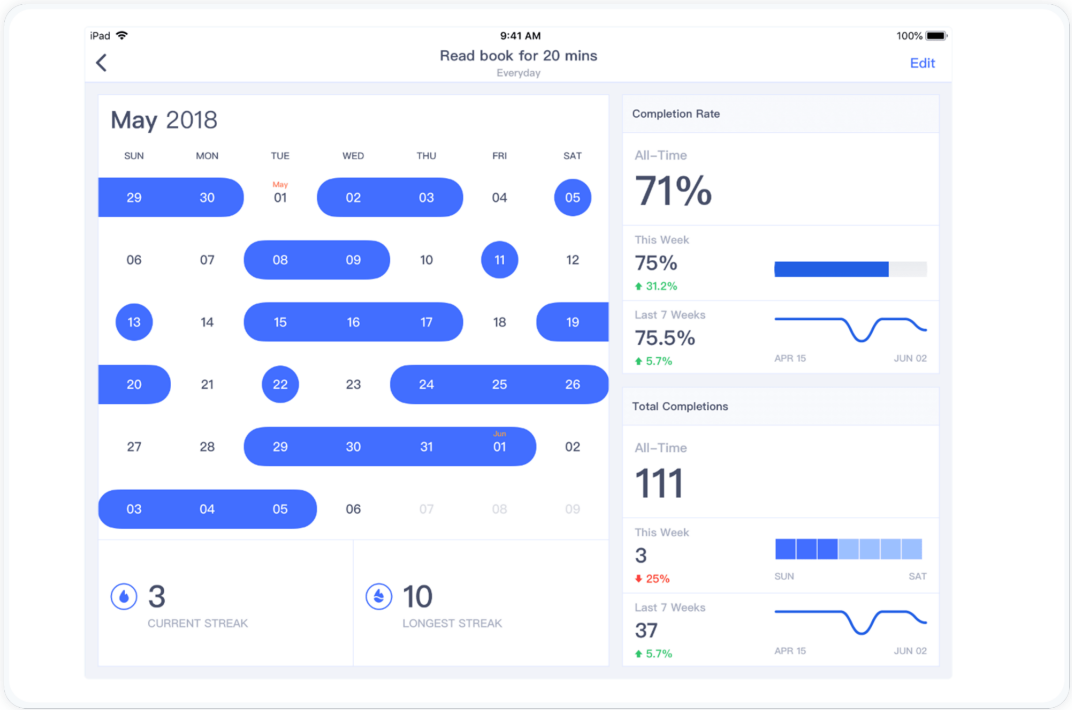


Рисунок 1 - Додаток Habitify

Додаток із доволі простою кольоровою гамою, проте привабливим інтерфейсом. Дане застосування являється досить гнучким. Він відстежує звички з різною частотою, наприклад, щодня або ж двічі на тиждень. У додатку наявна можливість отримувати у певну визначену годину доби, аби виконувати свої

звички протягом дня вчасно. Однією з особливостей цього додатка - це діаграми, які показують щоденні, щотижневі та щомісячні результати. У разі виникнення бажання переглянути щорічні результати - потрібно купити платну версію застосування. Крім того, у покращеній платній версії існує можливість архівування звичок, що дозволяє відмовитися від звичок на певний період часу та приступити до їх виконання згодом. До переваг даного додатку відноситься також його кросплатформність, адже він доступний і на Android, і на iOS, і як веб-додаток. Проте було виявлено наступний недолік у програмі: якщо користувач має бажання виконувати певну звичку декілька раз на день, він не матиме змогу вказати це у додатку і єдиним виходом із такої ситуації є дублювання цієї звички ще один раз власноруч.

2.Coach.me

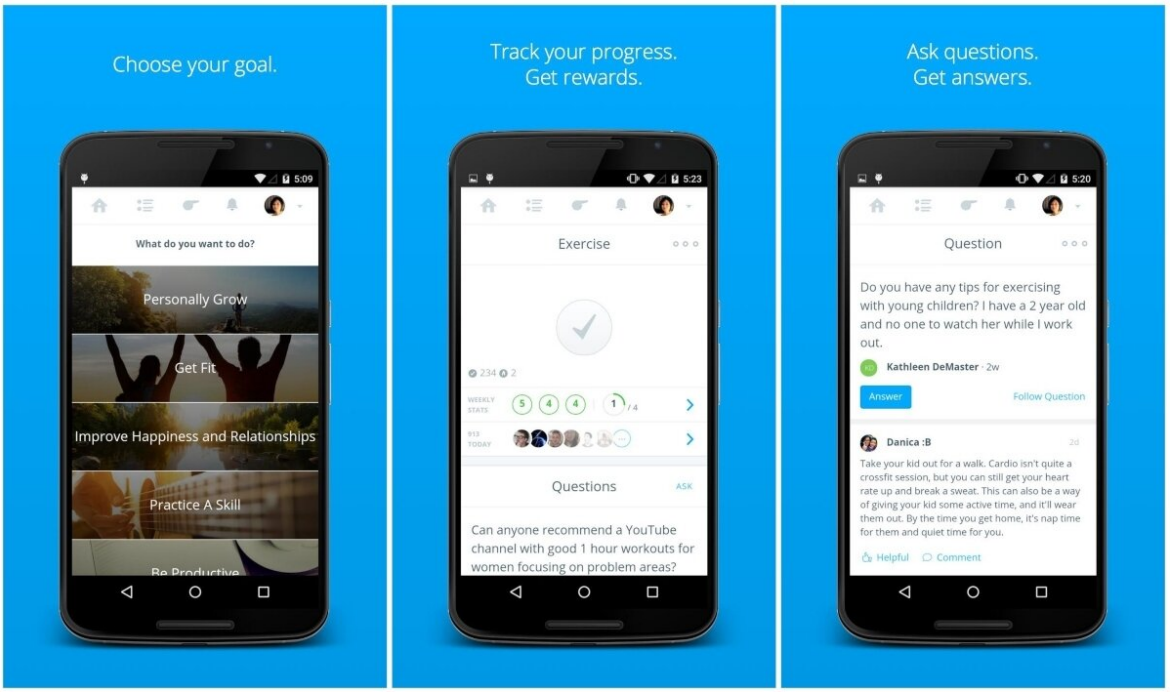


Рисунок 2 - Додаток Coach.me

Додаток для відстеження звичок, який окрім того, що допомагає зосередитись на щоденній практиці, надає доступ до професійних тренерів та спільноти, яка підтримує одного та надає поради. Для будь-якої поширеної звички, яку ви додаєте до свого списку, існує, ймовірно, обговорення, де ви можете зв’язатися з іншими користувачами платформи Coach.me. Тут ви можете поділитися ідеями

щодо того, як ефективно виконувати звичку, а також отримувати підтримку та заохочення, коли ви боретеся. Також існує можливість обрати собі тренера із потрібної категорії за певну плату та спілкуватись з ним один на один. Для того, аби не забувати про свої заплановані цілі - користувачам надходять сповіщення у додатку. У разі досягнення своєї цілі - Coach.me записує успіх користувача та повідомляє інших користувачів платформи, аби ті мали змогу привітати його з досягненням.

3. TickTick

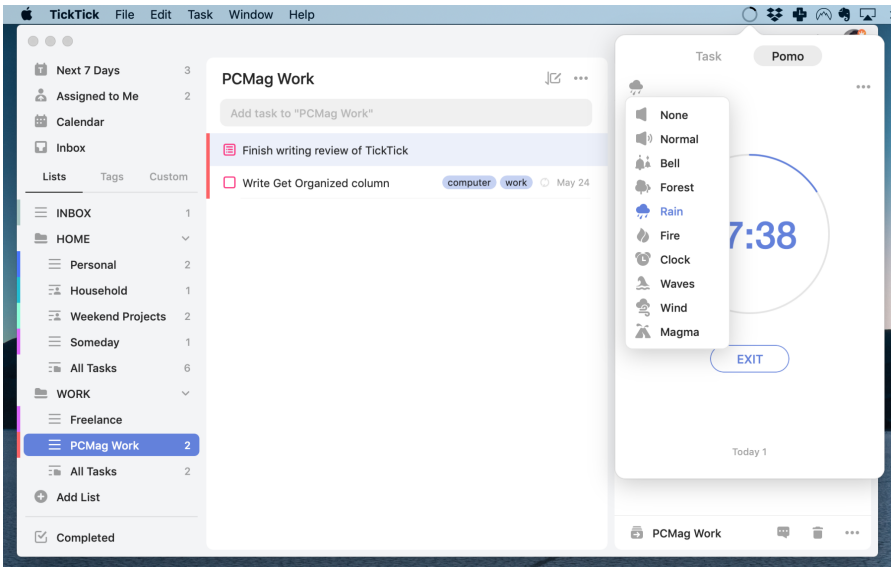


Рисунок 3 - Додаток TickTick

Додаток, який поєднує To-do list із можливістю відстеження звичок. Основний функціонал додатку - створення завдань(введення вручну на клавіатурі та за допомогою голосового введення), сортування їх за списками, визначення кожному завданню дедлайн та пріоритет, налаштування повторюваних завдань. Користувач також має змогу добавляти власні примітки та списки із підзадачами до кожного завдання. Для того, аби згрупувати певні завдання, існує можливість надати завданням власні теги. Аналогічно, як і у попередніх веб-трекерів, для того аби не забути за виконання завдання, можна встановити сповіщення, яке буде надсилатись користувачеві до закінчення терміну. На мою думку, даний варіант підійде тим, кому потрібен один інструмент для відстеження і списку справ, і звичок.

4. Strides

Рисунок 4 - Додаток Strides

Додаток, що пропонує різноманітні способи відстеження різних типів звичок. Основна функціональність набування нових навичок така ж, як у Habitify. Однак найголовніше, що Strides дозволяє відстежувати “середню” кількість для звички. Це корисно, якщо користувач не планує виконувати звичку у конкретні дні, але хоче дотримуватися певного середнього значення протягом дня, тижня, місяця, або ж року. Також у даному застосунку можна не лише створювати власноруч звичку, а й переглядати перелік певних готових шаблонів, обрати серед них ту, яка зацікавила і розпочати її виконання. Для того, аби переглянути хід виконання завдань, у додатку присутні індикатори, що відображають прогрес користувача.

1.3 Постановка задачі

У результаті ретельного аналізу даної предметної області та перегляду конкурентних існуючих аналогів розробки сформоване чітке уявлення про вимоги до системи, що повинна розроблятись. Необхідно створити веб-застосування, яке буде естетично привабливим для користувача та зрозумілим у користуванні. Варто зазначити, що у даному застосунку

неавторизовані користувачі матимуть змогу лише переглядати наявні виклики(челенджі), застосовувати фільтр та здійснювати пошук. Розглянемо, який саме функціонал буде доступним зареєстрованим користувачам.

Розробка веб-трекеру повинна задовольняти наступним функціональним вимогам:

- реєстрація та автентифікація користувачів;
- переглядати власні щоденні справи, звички, челенджі, які потрібно виконати впродовж сьогоднішнього дня.
- переглядати власний щоденний прогрес та кількість виконаних звичок та пройдених челенджів за весь час з періоду реєстрації;
- можливість загрузити фото профілю та, при бажанні, змінювати його;
- можливість додавати, видаляти та виконувати щоденні справи у свій To-do list.
- можливість створювати нову звичку власноруч або за допомогою заготовленого шаблона;
- можливість видаляти звичку із списку;
- можливість позначати виконаною або не виконаною звичку;
- у разі досягнення своєї цілі, тобто коли певна дія стає звичкою і виконується вже автоматично, існує можливість успішно завершити виконання даної звички;
- можливість створювати новий челлендж та видаляти його при бажанні;
- можливість переходити у спільний чат челенджу та спілкуватись з іншими користувачами, які теж беруть участь у даному випробуванні;
- переглядати всі наявні випробування;
- пошук та фільтрація челенджів та приєднання до бажаного.

1.4 Висновки до першого розділу

Отже, у першому розділі було здійснено аналіз сучасного стану питання, розглянуто різні способи трекінгу звичок, обґрунтовано актуальність створення даного веб-застосунку. Крім того, було проведено огляд існуючих додатків-аналогів з метою знаходження їхніх переваг та недоліків. Базуючись на

даних проведеного дослідження, було сформовано постановку завдання, яка містить детальний опис можливостей, що мають бути розробленні під час реалізації застосування.

РОЗДІЛ 2

2.1. Основні принципи розробки веб-застосувань

Клієнт-серверна архітектура - одна із архітектурних шаблонів програмного забезпечення, що є домінуючою концепцією у створенні веб-застосувань. Її основні компоненти:

- мережа, яка забезпечує взаємодію між серверами та клієнтами;
- набір серверів, що надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, що використовують сервіси, які надаються серверами;

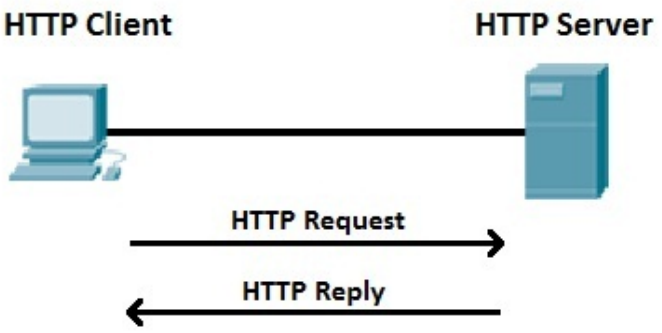
Сервери є незалежними один від одного, як і клієнти. Один сервер одночасно має змогу обробляти запити від різних клієнтів. Клієнти повинні знати про доступні сервери та можуть звертатись то до одного серверу, то до одного серверу, то до іншого. Прийнято вважати, що клієнти та сервери - це програмні модулі, які знаходяться у більшості випадках на різних комп’ютерах. У випадку, якщо і клієнтська і серверна частини знаходяться фізично на одному пристрої - то сервер називають локальним. Модель клієнт-серверної взаємодії визначається розподілом обов’язків між сервером та клієнтом. Компоненти інформаційної системи можна розділити на 3 основні шари:

1. Шар представлення - все, що пов'язано із взаємодією з користувачем;
2. Бізнес-логіка - прикладний рівень, який реалізує алгоритми реакції додатка на дії користувача, правила обробки даних;
3. Шар доступу до даних - рівень управління, який забезпечує зберігання, вибірку, модифікацію та видалення даних.

У клієнт-серверній архітектурі, в залежності від того, як розподілені та реалізовані вище наведені шари між сервером та клієнтом, існують 2 типи моделі взаємодії: тонкий та товстий клієнти. У тонкому клієнті вся логіка застосунку та управління даними зосереджена на сервері, а клієнтська програма відповідає лише за функції рівня представлення. У моделі товстого клієнта - обробка інформації та інтерфейс користувача зосереджені на клієнтській частині, тим часом як серверна частина лише керує даними. Зазвичай, клієнтом

виступає браузер користувача, а сервером - більш потужний комп’ютер з програмним кодом, що розташований або у хмарі, або локально.

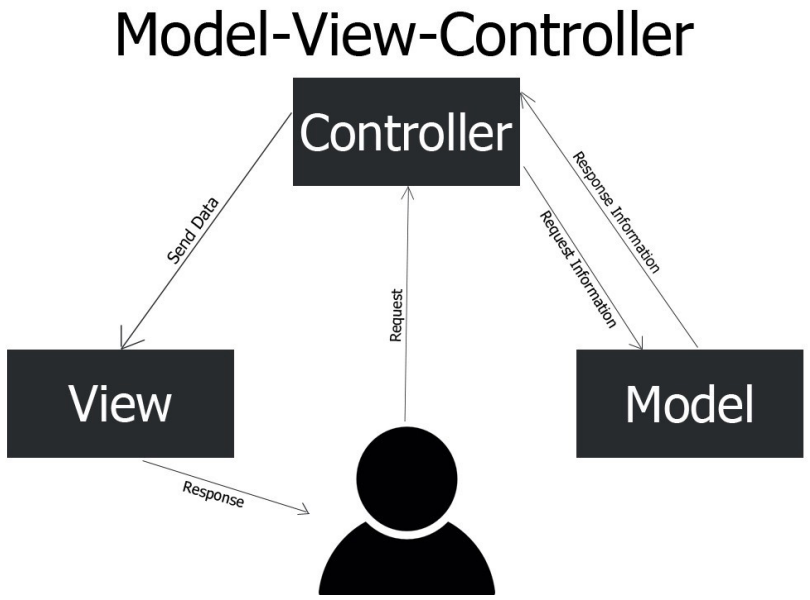
Для того, аби клієнт та сервер могли передавати один одному інформацію, існує протокол передачі даних по мережі на прикладному рівні моделі OSI HTTP(HTTPS). За допомогою HTTP-протоколу, можна відправити запит на сервер та отримати від нього відповідь. HTTP-запит складається з 3 частин: Starting line - відображає службові дані, такі як: один із можливих методів запиту(GET, POST, PUT, OPTIONS, HEAD, PATCH, DELETE, TRACE, CONNECT), шлях та версія протоколу передачі даних; Headers - опис параметрів повідомлення, наприклад хост, якому адресований запит та клієнт, який відправляє запит; Body - тіло повідомлення, яке є не обов'язковою частиною. У HTTP-протоколі передбачена велика кількість хедерів, використовуючи які можна налаштувати гнучке спілкування між клієнтом і сервером. Проте, з розвитком технологій і активним переміщенням персональних даних в інтернет довелося задуматися про те, як забезпечити додатковий захист інформації, яку клієнт передає серверу. В результаті з'явився протокол HTTPS, який синтаксично ідентичний протоколу HTTP, тобто використовує ті ж стартові рядки і заголовки. Єдині відмінності - додаткове шифрування і порт за замовчуванням 443, а не 80.



Крім того, під час розробки веб-застосувань варто використовувати певний шаблон проектування, як спосіб організації коду, який передбачає виділення блоків, що відповідають за вирішення різних задач. Одним із

найпопулярніших шаблонів проектування та розробки програмного забезпечення є MVC(Model-View-Controller), який передбачає розподіл даних додатку, користувацького інтерфейсу та керуючої логіки на окремі компоненти таким чином, що модифікація кожного компоненту може здійснюватись незалежно від інших компонентів. У даному шаблоні проектування наявні такі 2 компоненти:

- **Модель(Model)** - відповідає за дані, які зберігаються в базі даних і обробляються на сервері.
- **Представлення(View)** - HTML-шаблон, який повертає сервер після обробки запиту.
- **Контроллер(Controller)** - обробляє вхідні запити, зв'язує Представлення та Модель.



2.2. Класифікація сайтів

На даний момент у мережі Інтернет розташовано приблизно 1,89 млрд сайтів. Тим не менш, так як розвиток технологій стрімко зростає, як і попит на різний вид інформації, кількість інтернет-сторінок з кожним днем збільшується. Кожен із них належить одній або ж декільком категоріям, які визначаються згідно основної цілі веб-сторінки. Наразі однієї загальної класифікація не існує, проте можна виділити окремі категорії, такі як:

- **Business websites** - представлення певного бізнесу в мережі Інтернет. Основна мета створення такого сайту – розповсюдити інформацію про діяльність компанії серед якомога більшої кількості користувачів та зацікавити їх у придбанні/користуванні наявних товарів чи послуг, які надає дана організація.
- **Brochure and Catalogue websites** - це веб-сайти, основна ціль яких - реклама певного бізнесу, у якій вказується розташування основного комерційного сайту, перелік умов, відповідно яких надаються певні товари та послуги. Зазвичай дані електронні ресурси включають в себе відгуки інших користувачів на продукцію та тематичні дослідження. Основна відмінність веб-сайту Каталогу від Бізнес веб-сайту - відсутня можливість здійснення онлайн-оплати та придбання товарів одразу на сторінці.
- **Educational websites -** це веб-ресурс, основна ціль якого - надання навчальних матеріалів, або ж представлення про певний навчальний заклад, таких як школа чи університет, надаючи інформацію про деталі навчальної програми, її викладачів, розташування закладу та контактні дані.
- **Portal websites** - це веб-сайт, який виконує роль накопичення великої кількості інформації відносно інших веб-сторінок шляхом збереження посилань на них. Сторінки Портал сайту можуть посилатись абсолютно на різноманітні онлайн-сервіси, такі як новини, ігри, книги, форуми, блоги, тощо. Найвідоміші портальні веб-сайти з вбудованими пошуковими системами - Yahoo, AOL та MSN.
- **Crowdfunding websites -** розроблені для того, аби люди мали змогу звертатися з особистими проханнями на збір коштів необхідних потреб, наприклад таких як медичне обслуговування.
- **Informational websites -** сайти, основна мета яких - надання інформації на певну тему без можливості продажі курсів на основі власних знань.

- **Social media websites -** веб-ресурси, які призначені для того, аби надати можливість користувачам мережі Інтернет спілкуватись один з одним у приватних або групових чатах, ділитись новинами, обговорювати їх у коментарях і, загалом, соціалізуватись.
- **Entertainment websites** - сайти, основна мета яких - розважити користувача. На даних веб-сторінках часто розміщені ігри, комікси, фільми чи серіали, музика, тощо. Прикладами даної категорії є всесвітньо відомі сайти Youtube, Spotify, Netflix.
- **Blog websites -** веб-ресурс, який містить стрічку опублікованих новин певним користувачем, розміщених у хронологічному порядку. Дані публікації можуть містити як великий обсяг інформації, так і зображення.

Згідно цих даних, можна зробити висновок, що створений впродовж даної курсової роботи веб-застосунок має спільні характеристики з Social media website та Blog website, так як у ньому буде присутня можливість комунікації та опублікування створених челенджів користувача у стрічці.

2.3 Висновки до другого розділу

У другому розділі курсової роботи було розглянуто основні принципи розробки веб-застосувань, а саме основні компоненти клієнт-серверної архітектури, принципи її роботи, різницю між тонким та товстим клієнтом. Було описано протокол передачі даних по мережі на прикладному рівні моделі OSI HTTP(HTTPS), його структуру та застосування. Також, було з'ясовано, що для того, аби розподілити дані додатку, користувацького інтерфейсу та керуючої логіки на окремі компоненти варто використовувати паттерн MVC, який було вирішено розробляти у даному застосуванні.

Крім того, у цьому розділі було досліджено, які існують види веб-сайтів, і за допомогою класифікації з’ясовано, що розроблений додаток матиме спільні характеристики із типами Social media website та Blog website.

РОЗДІЛ 3

3.1 Аналіз технічного завдання

У розробленому веб-застосунку наявні наступні групи користувачів: зареєстровані користувачі, не зареєстровані та адміністратори.

Технічні вимоги користувачів до функціональності системи:

Незареєстрований користувач

- можливість зареєструватись та пройти аутентифікацію для входу в систему;
- можливість перегляду публічних челенджів та здійснення фільтрації.

Зареєстрований користувач

- можливість перегляду особистих даних, щоденного прогресу та редагування власного фото профілю;
- перегляд справ, челенджів та звичок, які заплановані на сьогодні;
- перегляд справ, які не були виконані вчасно, тобто були пропущені та можливість їх виконання;
- перегляд челенджів, які заплановані на майбутнє;
- перегляд раніше пройдених звичок та челенджів з можливістю пройти їх ще раз;
- перегляд власної статистики за поточний тиждень, місяць, чи останніх 3 місяці;
- створення, видалення та виконання нової запланованої справи;
- можливість перегляду умов доданого челенджу;
- створення, видалення та виконання умов нового челенджу;
- можливість спілкуватись з іншими користувачами веб-ресурсу, які беруть у тому самому публічному челенджі;
- створення власноруч, або за допомогою заготовленого шаблону нової звички;
- можливість відправляти власну звичку на огляд адміністратору для підтвердження/відмову добавлення її у заготовлені шаблони звичок;
- можливість позначати звичку виконаною, або невиконаною;
- можливість видалення звички у разі зникнення бажання її виконувати;
- можливість позначати звичку набутою/досягнутою;

- можливість пошуку та фільтрації раніше створених челенджів інших користувачів;
- можливість доєднатись до чужого челенджу або зробити його собі приватним;
- у разі створення власноруч челенджів, можливість їх перегляду та редагування і видалення у випадку, якщо початок челенджу ще не відбувся;

Адміністратор

- можливість переглядати надіслані користувачами запити на підтвердження/скасування добавлення створеної звички у заготовлені шаблони;
- можливість підтверджувати запит користувача;
- можливість скасовувати запит користувача;

Специфікація вимог до даних:

1. **Користувач:**

Про користувача повинна зберігатись наступна інформація: його нікнейм, електронна пошта та пароль, фото профілю, його справи, звички та челенджі.

2. **Справа**

Справа - це одноразова дія, яка не повинна ставати звичкою, проте має бути виконаною до певного терміну. Для того, аби коректно реалізувати справи користувача, потрібно зберігати унікальний ідентифікатор користувача, який запланував дану справу, її назву, дедлайн, та статус виконання.

3. **Челендж**

Челендж - це кількаденний виклик, який може проходити публічно, тобто групою користувачів, або приватно, тобто окремо від інших. Для збереження челенджу в базі даних потрібно зберігати наступну інформацію про нього: назву, ціль проходження челенджу, його аватарка, адміністратор(користувач, що його організовує), день старту, тривалість виклику(кількість днів), рівень

приватності, умови челенджу, яких має дотримуватись кожен учасник, опис, в якому вказувати про що і для кого це випробування, та причину, яка може зацікавити інших у проходженні.

4. Звичка

Звичка - це дія, яку користувач бажає зробити для себе автоматичною. У даному веб-ресурсі існує 2 головних типи звички: та, яка базується на повторенні певної дії (наприклад, медитувати двічі на день) і та, що базується на повторенні певної кількості під час виконання дії(наприклад, медитувати 30 хв впродовж дня). Для зберігання в базі потрібна наступна інформація: назва, її ціль, її тип, період часу.

Корпоративні обмеження цілісності даних:

- Нікнейм та електронна пошта користувача повинні бути унікальними. Нікнейм користувача має містити більше 3 символів. Пароль повинен містити від 8 до 15 символів, включаючи обов'язково 1 велику літеру, 1 малу, 1 цифру та 1 спеціальний символ.
- При створенні нової справи поля назва і термін здачі є обов'язковими та повинні бути унікальними. Але у різних користувачів можуть бути однакові справи(з однаковою назвою та дедлайном).
- При створенні нового виклику, користувач повинен обирати ціль даного випробування з випадаючого списку, де будуть перечислені усі раніше створені цілі. У випадку, якщо користувач не знайшов бажаної мети - має можливість ввести свою власну. Крім того, дата початку челенджу повинна бути не меншою за сьогоднішню дату.
- При створенні нового виклику, такі поля як “назва”, “ціль”, “опис”, “день старту”, “тривалість”, “приватність”, “кроки виконання” є обов’язковими. Також, назва випробування повинна бути унікальною і поле “кроки виконання” має містити від 1 до 10 можливих умов, яких бажано дотримуватись під час проходження випробування кожному учаснику.
- Тривалість кожного виклику повинна бути позитивним числом.

- При створенні нової звички усі поля є обов’язковими. Поле “назва” має бути унікальним. Значення у полі “дата початку виконання” звички повинна бути не меншою за сьогоднішню дату. Поле “тип звички” повинно бути обраним із списку з 2 елементів: повторювати дію або ж повторювати кількість для певної дії. Поле “період часу” також повинно бути обраним із випадаючого списку з 3 елементів: щодень, щотижня, чи щомісяця.
- Якщо челендж минув, то користувач може доєднатись до цього челенджу лише зробивши його для себе приватним.
- Якщо челендж триває зараз, або ж запланований у майбутньому, користувач має можливість приєднатись до існуючого, або ж зробити його собі приватним, спланувавши його на певну дату.
- У фільтрації випробувань повинен відображатись перелік цілей, які наявні у хоча б одному з випробувань.
- Якщо челендж ще не розпочався, адміністратор челенджу має можливість:
    a) редагувати всі поля, крім назви та цілі створеного виклику;
    b) видаляти весь челендж, при цьому оновлення повинні відбуватись у всіх користувачів, які записались на його проходження і для яких він є публічним.

<u>Вимоги до надійності:</u>

- Неавторизований користувач не повинен мати доступу до всіх сторінок, крім сторінки реєстрації та логіну.
- Кожен зареєстрований користувач повинен бачити лише власний профіль з особистими планами, звичками та челенджами.
- Кожен зареєстрований користувач повинен бачити лише власну статистику.
- Користувач може зайти в чат челенджу лише тоді, коли він доєднався до нього та випробування було розпочато.

- Сторінка з створеними раніше челенджами повинна бути доступна лише для тих користувачів, які самостійно створювали виклики.
- При спробі переходу на неіснуючі посилання, потрібно переадресовувати користувачів на сторінку 404.

Користувацький інтерфейс:

Для того, аби усім користувачам було комфортно користуватись даним ресурсом, варто звернути увагу на розроблюваний інтерфейс. Перш за все, він повинен бути інтуїтивно зрозумілим для того, аби не виникало питань, як саме потрібно користуватись веб-сайтом. По-друге, він також має бути естетично привабливим та компактним, де не потрібно буде шукати заплановані на сьогодні справи, челенджі, чи звички. Проаналізувавши перелік функціоналу, який має бути виконано, було обрано розмістити меню веб-сайту вверху вздовж екрану для легкої навігації по існуючих сторінках веб-трекеру. На сторінці профілю користувача розміщені його основні завдання на сьогодні, розподіливши їх на 3 колонки для полегшення візуалізації запланованої роботи: Справи, Челенджі, Звички. Відносно кольорової гами, так як загальна концепція додатку - бути працелюбним та наполегливим “мов бджілка” було обрано 2 основні кольори - жовтий(#fdd32a) та чорний(#000000). Веб-сайт повинен відображатись як і на різних версіях браузера та операційної системи, так і на різних типах пристроїв(мобільний телефон, планшет, комп’ютер).

3.2 Обґрунтування вибору засобів розробки

Для того, аби створити повноцінний веб-застосунок потрібно обрати технології для розробки інтерфейсної та серверної частин. На даний момент, існує великий перелік можливих інструментів, які спрощують написання веб-додатків та дозволяють використовувати вже готові рішення. Розглянемо детальніше обрані технології для Front-end та Back-end розробки.

Клієнтська частина

Інтерфейс користувача буде розроблено, застосовуючи 3 базові мови програмування у веб-розробці: JavaScript, HTML та CSS.

HTML(HyperText Markup Language) - мова розмітки , яка взаємодіє з веб-браузером та диктує структуру кожної сторінки у Всесвітній павутині. Її основне призначення - повідомляти браузеру, як інтерпретувати дані для формування елементів на веб-сайті.

CSS(Cascading Style Sheets) - надає можливість описати спосіб відображення елементів HTML на пристрої користувача. За допомогою CSS можна визначати стиль шрифту, інтервали між абзацами, встановлювати фонові зображення, використовувати анімації, тощо. Перевага даної мови в тому, що її легко вивчити та зрозуміти, а також вона дозволяє адаптувати веб-сторінки до різних типів пристроїв, залежно від їх розмірів.

JavaScript - мова програмування, яка часто використовується під час розробки веб-застосувань. Основний функціонал цієї мови відповідає за динамічну модифікацію HTML сторінок та CSS за допомогою DOM API.

Для того, аби спростити обхід та маніпулювання документами HTML та обробку подій браузера використовувується JavaScript бібліотека JQuery, яка надає інструментарій із великою кількістю AJAX(Asynchronous JavaScript and XML) методів, які дозволяють завантажувати дані із сервера без перезавантаження сторінки. Крім того, аби створити візуально привабливий інтерфейс для користувача було обрано фреймворк Bootstrap. Переваги використання даного фреймворку:

- наявна можливість використовувати готові теми та шаблони дизайну як вихідну точку;
- загальнодоступний(open-source), легкий для використання та встановлення;
- наявність “responsive grid system”, що дозволяє адаптувати дизайн до девайсів різних розмірів;
- підтримка сумісності з різними браузерами.

Серверна частина

Розглядалось 3 варіанти для написання серверної частини веб-додатку: Node JS , PHP, Python. Переваги, через які було обрано Node JS, яка реалізована за допомогою мови програмування JavaScript:

- Node.js добре підходить для розробки веб-додатків, що реагують на дії користувача в режимі реального часу;
- підтримка NPM з багатьма модулями – на відміну від php, npm встроєний за замовчуванням, а менеджер пакетів Composer, що присутній у php прийдеться вбудовувати самостійно;
- корпоративна підтримка – ряд компаній, таких як IBM, Microsoft, PayPal , організували фонд NodeJS, що сприяє розвитку його основних інструментів;
- Node.js є другим найбільш популярним сховищем GitHub, що означає, що він має дуже активну екосистему і спільноту розробників, які завжди готові допомогти;
- був використаний у написанні таких застосувань як Netflix , PayPal, Trello, LINKEDIN, Yahoo, Uber, Medium, Yandex, Ebay, що означає, що Node JS має досить великі перспективи та буде розвиватись і надалі;
- на відміну від Python, який є більш універсальним, адже його використовують як у веб-розробці, так і в інших галузях, таких як Data Science, Machine Learning, Node Js більше “заточений” під веб-розробку.

Для реалізації основного функціоналу серверної частини було використано наступні фреймворки та бібліотеки:

- **Express фреймворк** - надає методи, які дозволяють вказати, яка функція та URL шаблон, викликається для конкретного HTTP запиту (GET, POST, SET).
- **Mongoose** - відповідає за підключення бази даних та представлення об’єктів MongoDB у коді.
- **Formidable** - використовується для обробки вхідних даних та завантаження файлів.

- **EJS(Embedded Javascript Templating)** - шаблонізатор, що надає можливість вводити дані із сервера на стороні клієнта, тим самим дозволяючи генерування HTML сторінок з мінімальним кодом.
- **Jsonwebtoken** - відповідає за аутентифікацію користувачів.
- **Socket.io** - створює двонапрямлений(bi-directional) зв’язок між клієнтом та сервером, що надає можливість спілкування користувачів у реальному часі.
- **Body-parser -** бібліотека, основна ціль якої - парсити вхідні дані до формату json.
- **Cookie-parser -** бібліотека, яка відповідає за парсинг cookie, що прикріплені до запиту з сторони клієнта.

Бази даних

У даному веб-застосунку буде використовуватись нереляційна база даних MongoDB через наступний ряд переваг:

- перетворення та відображення об'єктів програми в об'єкти бази даних не потрібне;
- структура окремого об'єкта чітка та зрозуміла;
- жодних складних об’єднань;
- глибокі можливості запитів - MongoDB підтримує динамічні запити в документах, використовуючи мову запитів на основі документів;
- MongoDB легко масштабувати.

Для того, аби мати можливість зручно та швидко керувати об’єктами бази даних було використано MongoDB Compass.

3.3 Опис розробки додатку

Веб-застосування розроблялось за наступним планом дій:

1. Зроблено верстку HTML-сторінок веб-сайту з розширенням .ejs, підключення до них необхідних CSS-файлів, Bootstrap, JQuery.
2. Інсталювання усіх необхідних модулів в менеджері пакетів NodeJS npm за допомогою команди npm install <nameModule>. Після її виконання

створюється папка node_modules, яка містить в собі усі завантажені залежності, перелік яких відображається також у файлі package.json у блоці dependencies/devDependencies.

3. Було створено базу даних “habit_tracker” в MongoDB Compass.
4. У кореневій папці проекту було створено папку config, у котрій знаходиться файл keys.js, що містить шлях до створеної у попередньому пункті бази даних.
5. Далі для того, аби покращити структуру коду та не писати кілька раз одні і ті ж самі залежності, які мають підключатись у багатьох файлах, створено файл container.js, у якому, завдяки модулю dependable, для кожної залежності, доданої раніше у загальний масив, застосовується метод require().
6. Наступний крок - створення головного файлу запуску додатку - app.js, у якому буде підключено базу даних за допомогою модуля mongoose та реалізовано HTTP сервер за допомогою модуля http наступним чином:

```
const mongoose = require('mongoose');
const express = require('express');
const container = require('./container');
container.resolve(function(users,dashboard,group,_){
 const db = require('./config/keys').MongoURI;
mongoose.connect(db, { useNewUrlParser: true, useUnifiedTopology: true })
.then(() => console.log("Connected to MongoDB successfully!"))
 .catch(err => console.log(err));
const app = express();
 const server = http.createServer(app);
     server.listen(process.env.PORT || 3000, function(){console.log('Listening on port 3000'); });
});
```

Після того, як створили базовий функціонал серверу, потрібно доналаштувати проміжне програмне забезпечення(middleware) для додатка.

7. Наступний етап - створення на клієнтській частині функціоналу, що відповідає за відправку запитів GET (на отримання даних) та POST(на зміну даних).
8. Далі було створено файли, які приймають різноманітні запити від користувачів та надсилають відповідь із серверної частини на клієнтську. Для кращої побудови і структуризації коду, аби не писати всі запити у одному файлі, їх було розподілено на 3 основні: ті, які приймають запити під час реєстрації та логіну; ті, що приймають запити, що пов’язані з чатами користувачів; ті, які виконують запити відносно усіх справ, челенджів та звичок.
9. Після того, як із серверу було відправлено відповідь із певним статусом, на клієнтській частині повинен бути реалізований функціонал, що очікує відповідь та реагує на результат(наприклад, перенаправляє на сторінку профілю при вдалому логіні, або ж змінює частину коду HTML сторінки без перезавантаження).
10. Останній, проте не менш важливий пункт - мануальне тестування , знаходження недоліків та відлагодження функцій програми.

3.4 Створення об’єктів та розробка головного функціоналу

Структура створеного веб-додатку наведена на рисунку 1. У кореневій папці проекту знаходяться наступні файли:

- .env - файл, у якому зберігається згенерований за допомогою модуля bcrypt токен доступу(ACCESS_TOKEN), що використовується для генерації токена користувача під час успішної аутентифікації;
- app.js та container.js - файли проекту, основний функціонал яких був описаний у попередньому пункті(3.3);

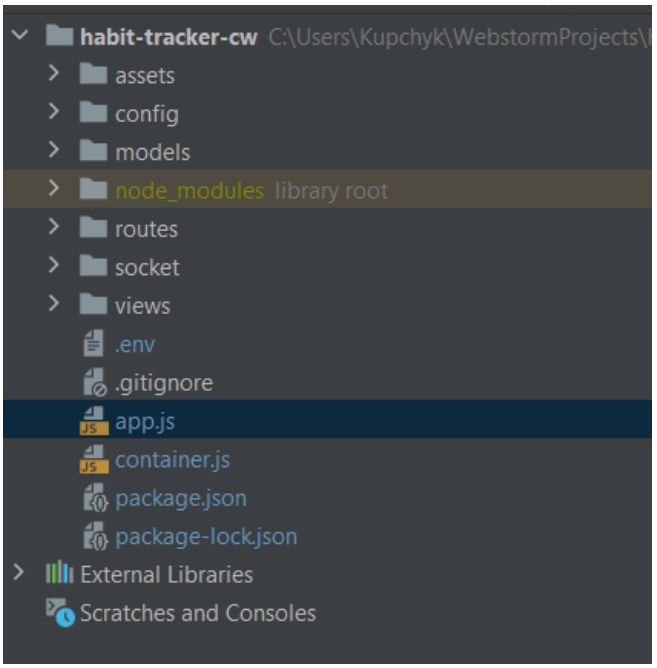


Рисунок 5 - Структура проекту

Директорія **assets** містить у собі інформацію, яка поділена на 4 підкатегорії: **css, js, img**(директорія, у якій знаходяться всі зображення, які використовувались під час розробки дизайну сторінок), **uploads**(директорія, яка використовується для збереження фото, які завантажує користувач під час зміни аватарки профілю, або ж під час додавання фото протягом створення челенджу).

Директорія **config** містить 1 файл keys.js, у якому знаходиться посилання для встановлення зв'язку з MongoDB.

Директорія **models** зберігає у собі файли, які описують схеми моделей, що зберігаються в базі даних.

Директорія **routes** містить 3 файли: users.js, group.js, dashboard.js, які відповідають за отримання запитів зі сторони клієнта та надсилання згенерованої відповіді. У файлі users.js(рис. 2) знаходяться набір роутів з методами GET та POST, які відповідають за реєстрацію, аутентифікацію та вихід з системи користувача. Під час реєстрації дані користувача валідуються на сервері, перевіряючи, чи нікнейм користувача довший, ніж 3 символа та чи пароль відповідає раніше наведеним вимогам. Якщо так, відбувається перевірка на унікальність його нікнейму та електронної пошти серед інших зареєстрованих користувачів. У разі успішної валідації - введений пароль

шифрується за допомогою модуля bcrypt, аби захистити особисті дані, зберігає нового користувача у базу даних та перенаправляє на сторінку аутентифікації. Для того, аби успішно залогінитись у систему, потрібно ввести власний нікнейм та пароль, що згенерує унікальний JWT токен користувача, збереже його у файли cookies та перенаправить на сторінку власного профілю. У випадку невдалої реєстрації або ж аутентифікації з серверу надсилаються причини невалідних даних та висвічуються відповідно на сторінках.

```
SetRouting: function (router) {
    router.get('/login',this.gtLogin);
    router.get('/register',this.gtRgistr);
    router.get('/logout',this.gtLogout);
    router.post('/register', this.postRgistr);
    router.post('/login',this.postLogin);
},
```

Рисунок 6 - users.js

У файлі group.js(рис. 3) - набір методів, відповідальних за генерування сторінки чату челенджу, збереження нових повідомлень від користувачів, які доєднались до чату та відправили повідомлення, та зміну статусу окремого кроку челенджу, які висвітлюються у лівій боковій панелі. Для того, аби коректно генерувати сторінку чату випробування, у шляху передається параметром його унікальний ідентифікатор, що був самостійно згенерований за допомогою MongoDB.

```
SetRouting: function(router){
    router.get('/challenge-chat/:id', authenticateToken, this.groupPage);
    router.post('/challenge-chat/:id', authenticateToken,this.groupPostPage);
    router.post('/statusChanged',authenticateToken, this.changeStatus);
},
```

Рисунок 7 - group.js

У файлі dashboard.js(рис. 4) - знаходяться усі можливі шляхи, що відповідають за перегляд, створення, видалення, редагування та виконання раніше створених справ/звичок/випробувань. Крім того, присутні роути, що відповідають за генерування відфільтрованих челенджів за ціллю, або ж за полем вводу. Також

даний файл відповідає за генерування пройдених челенджів, звичок, завантаження нового зображення користувачем та його збереження.

```
SetRouting: function (router) {
    router.get('/dashboard', authenticateToken, this.gtDashboard);
    router.get('/countdown', authenticateToken,this.gtTimr);
    router.get('/add-new-challenge', authenticateToken,this.gtAddChallenge);
    router.get('/challenges', authenticateToken,this.getChallenges);
    router.get('/filtered', authenticateToken, this.getResults);
    router.get('/filteredInput', authenticateToken, this.getResultsInput);
    router.get('/template-habits', authenticateToken, this.getTemplateHabits);
    router.get('/created-challenges', authenticateToken, this.getCreatedChallenges);
    router.get('/edit-challenge', authenticateToken, this.editChallenge);
    router.get('/statistics', authenticateToken,this.getStatistics);
    router.get('/challenge/:id', authenticateToken,this.getChallengePage);
    router.get('/doneChallenges', authenticateToken,this.getMyDoneChallenges);
    router.get('/doneHabits', authenticateToken,this.getMyDoneHabits);

    router.post('/countdown', authenticateToken,this.postTimr);
    router.post('/add-new-challenge', authenticateToken, this.postChallenge);
    router.post('/edit-challenge', authenticateToken, this.editChallengePost);
    router.post('/delete-challenge-admin', authenticateToken, this.deleteChallengeAdmin);
    router.post('/add-habit', authenticateToken, this.addNewHabit);
    router.post('/count-habit',authenticateToken, this.countHabit);
    router.post('/uploadFile', this.uploadFile);
    router.post('/savePhoto', authenticateToken, this.savePhoto);
    router.post('/add-challenge', authenticateToken, this.addChallenge);
    router.post('/program', authenticateToken,this.postProgram);
    router.post('/delete-program', authenticateToken,this.postDelProg);
    router.post('/delete-challenge',authenticateToken, this.postDelChallenge);
    router.post('/delete-habit',authenticateToken,this.postDelHabit);
    router.post('/habitStatus',authenticateToken,this.postHabitStatus);
},
```

Рисунок 8 - dashboard.js

Для того, аби заборонити доступ до сторінок, які мають бути доступними лише для автентифікованих користувачів, використовується метод **authenticateToken**(рис. 5), який реалізований наступним чином: із файлу cookies потрібно спробувати витягнути значення за ключем “jwt” (так як при логіні згенерований токен додавався саме з таким ключем). Якщо токену не існує - повернеться статус 401, що означатиме, що користувач не авторизований у системі. У іншому випадку, потрібно віднайти юзера за даним токеном і якщо не існує наявних помилок - встановити у значення req.user знайденого

користувача. Якщо наявні помилки, тобто токен, який був знайдений, не є валідним, повертається статус 403, який означає, що доступ до сторінки заборонений.

```
require('dotenv').config();
const jwt = require('jsonwebtoken');

function authenticateToken(req, res, next) {
    const token = req.cookies.jwt;
    if (token == null)
        return res.sendStatus( code: 401);

    jwt.verify(token, process.env.ACCESS_TOKEN_SECRET,  options: (err, user) => {
        if (err) return res.sendStatus( code: 403);
        req.user = user;
        next();
    })
}
```

Рисунок 9 - Перевірка аутентфікації

У директорії **socket** знаходиться файл groupchat.js, який відповідає за встановлення сокетного зв'язку на серверній частині створеного додатку, який потрібен для реалізації можливості чату у реальному часі.

Директорія **views** містить у собі файли, які відображають користувацький інтерфейс.

3.5 Опис бази даних

Для того, аби відповідати усім вимогам з технічного аналізу завдання, було створено базу даних **habit_tracker**, структура таблиць якої зображена на рисунку 6.

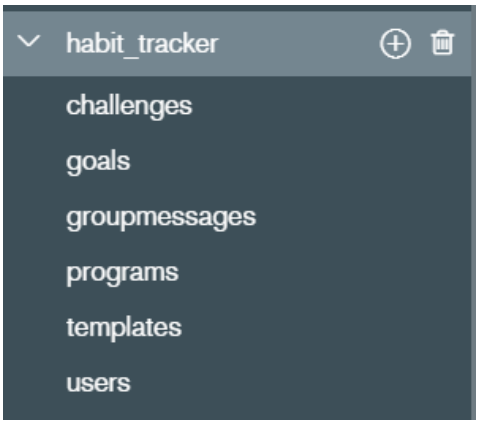


Рисунок 10 - Структура бази даних в MongoDB Compass

Колекція **challenges** - містить у собі об'єкти, які відповідають моделі із директорії models/Challenge.js. Для того, аби реалізувати схему у базі Монго, потрібно використати метод mongoose.Schema(Object), де Object - об’єкт з усіма полями та їх обмеженнями, наступним чином:

```
const ChallengeSchema = new mongoose.Schema({
   user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
   image: { type: String, required:true, default: 'challengeAccepted.jpg'},
   goal:{type: String, required: true},
   days:{type:Number, required:true},
   name:{type: String,unique:true, required: true},
   description:{type: String, required: true},
   startDate:{type:Date, required:true},
   why:{ type: String, required: false},
   steps: [{type: String}],
   users:[{
      userId:{ type: mongoose.Schema.Types.ObjectId, ref: 'User'},
      status:{ type:String}
   }]
});
```

Дана схема створена для того, аби зберігати усі публічні челенджі у одній колекції. Вона містить усі поля, які вимагаються у технічному аналізі завдання і відповідає усім корпоративним обмеженням цілісності даних. Крім того, у ній також знаходиться масив об'єктів users, кожен з яких складається з унікального ідентифікатора користувача і статуса виконання челенджу(“хоче виконати”,“виконав”, “відмовився”). Він створений для того, аби мати перелік усіх користувачів, що підписались на виконання даного випробування. Завдяки такій структурі адміністратор челенджу(користувач, хто його створив), може оновлювати дані або ж видаляти до початку старту, тим самим оновивши дані і у інших користувачів.

Колекція **goals** - містить у собі об'єкти, які відповідають моделі із директорії models/GoalChallenge.js.

```
const GoalSchema = new mongoose.Schema({
   name: {type: String, unique:true, required: true}
});
```

Схема містить у собі унікальний перелік усіх цілей, які були раніше створені для челенджів. Дана схема потрібна для того, аби легше і швидше знаходити всю множину цілей на сторінці додавання нового челенджу, яка не містить у собі повторюваних значень.

Колекція **groupmessages -** містить у собі об'єкти, які відповідають моделі із директорії models/groupmessage.js.

```
var groupMessage = mongoose.Schema({
   sender: {type: mongoose.Schema.Types.ObjectId, ref: 'User'},
   body: {type: String},
   challengeId: {type:  mongoose.Schema.Types.ObjectId, ref:'Challenge'},
   createdAt: {type: Date, default: Date.now}
});
```

Схема зберігає об'єкти, які містять інформацію групових повідомлень з чатів. У ній зберігається унікальний ідентифікатор челенджу, час відправки повідомлення, унікальний ідентифікатор відправника і сам текст створеного месседжу.

Колекція **programs -** містить у собі об'єкти, які відповідають моделі із директорії models/Program.js.

```
const ProgramSchema = new mongoose.Schema({
   user: {type: mongoose.Schema.Types.ObjectId, ref: 'User'},
   progName:{type:String, required: true},
   deadline:{type: Date, required: true},
   progStatus:{type: Boolean, required:true}
});
```

Схема містить у собі об'єкти, які відповідають за збереження справ користувача (To-do). У ній зберігається інформація про назву справи, хто її має виконувати, кінцевий термін здачі та статус виконання.

Колекція **templates** -містить у собі об'єкти, які відповідають моделі із директорії models/Template.js.

```
const TemplateSchema = new mongoose.Schema({
   templateName:{type:String, required:true, unique:true},
   habits: [{
      habitName: {type: String, required:true, unique:true, sparse: true},
      goal : {type:Number, required:true},
      timePeriod:{type:String, required:true},
      habitType:{type:String, required:true},
      goalType:{type:String, required:true}
   }],
});
```

Схема відображає інформацію, про шаблони звичок. Вона містить такі поля, як назва шаблону, що повинна бути унікальною та масив звичок, що належать до цього шаблону.

Колекція **users -** містить у собі об'єкти, які відповідають моделі із директорії models/User.js.

```
const UserSchema = new mongoose.Schema({
   name: { type: String, required: true, unique: true},
   password: {type: String, required: true},
   email: {type: String, required: true, unique: true},
   userImage:{ type:String, required:true, default: 'default.png'},
   challenges: [{
      id: { type: mongoose.Schema.Types.ObjectId, ref: 'Challenge'},
      name: {type: String, default: ''},
      steps: [
```

```
        day:{type:Date,required:true},
        nameStep: {type: String},
        statusStep : {type:Boolean}
      }],
      startDate:{type:Date, required:true},
      finishDate:{type:Date, required:true},
      goal:{type: String, default: ''},
      privacy:{type:String, required:true},
      status:{type:Boolean,required:true},
   }],
   habits:[{
      habitName: {type: String, required:true, unique:true, sparse:true},
      goal : {type:Number, required:true},
      timePeriod:{type:String, required:true},
      habitType:{type:String, required:true},
      goalType:{type:String, required:true},
      startDate:{type:Date, required:true},
      status:{type:Boolean, required:true},
      days:[{
        day:{type:Date,required:false, unique:true, sparse: true},
        startDay:{type:Date,required:false},
        finishDay:{type:Date,required:false},
        countGoal:{type:Number,required:true}
      }]
   }]
});
```

Дана таблиця зберігає у собі нікнейм та емейл, які є унікальними за допомогою властивості unique, хеш паролю, аватар профілю, який має значення за замовчуванням за допомогою властивості default, яке використовується у випадку, коли користувач не надав свого власного зображення, масив челенджів та масив його звичок. Масив challenges створений для того, аби зберігати інформацію про всі челенджі, які завершив Юзер, лише розпочав, або ж

планується розпочати у майбутньому. У ньому містяться дані про назву виклику, дату старту, дату фінішу, яка вираховується в залежності від тривалості челенджу, до якого приєднується користувач, ціль, приватність(публічний або приватний), статус виконання(true - виконаний, false - невиконаний) та масив кроків, який формується з об'єктів, кожен з яких має призначений день виконання, особисту назву і статус. Масив усіх кроків містить у собі усі дні, починаючи від старту випробування, завершуючи датою його фінішу.

Відносно масиву усіх звичок користувача, він створений для того, аби аналогічно зберігати усі звички, які є виконаними, виконуються, або ж є запланованими. У ньому містяться дані про назву звички, її ціль, тип цілі(кількість раз, годин, хвилин,тощо), тип звички, день старту виконання, період часу(щодень, щотижня, щомісяць), статус прогресу та масив днів. У даному масиві, на відміну від попереднього з випробуваннями, не існує кінцевої дати виконання звички, адже, проаналізувавши предметну область і зясувавши, що одна і та сама звичка може засвоюватись різними людьми кардинально різний проміжок часу, було вирішено самостійно надати можливість користувачеві позначати певну звичку досягнутою, або ж відмовлятись від її виконання. У разі відмови - звичка буде видалятись із бази даних повністю, у разі досягнення - змінювати поле status = true. Крім того, так як існує певний період часу, за який повинна звичка повторюватись і база Монго має можливість зберігати у одній колекції документи з різними полями, було вирішено створити об'єкт масиву days наступним чином: поле day, що є необов'язковим та буде використовуватись лише для звичок, які повинні виконувати щоденно; поля startDay і finishDay, які теж є необов'язковими і використовуються для звичок, що є щотижневими чи щомісячними. Дана структура була обрана для того, аби мати можливість коректно розраховувати поле countGoal, яке, в залежності від періоду виконання, має обнулятись щоденно/щотижнево/щомісячно.

3.6 Інструкція користувача

Для того, аби увійти на сайт, користувачу потрібно зареєструватись, ввівши власний нікнейм, пароль та електронну пошту(рис. 10). Після вдалої реєстрації його перенаправить на сторінку логіну(рис. 12), де йому знадобиться ввести особисті дані для отримання доступу до веб-трекеру. У разі некоректних реєстраційних даних або ж аутентифікаційних - користувачу відображатимуться наявні проблеми та у полях вводу зберігатимуться його попередні ввідні дані(рис.11 та рис.13).

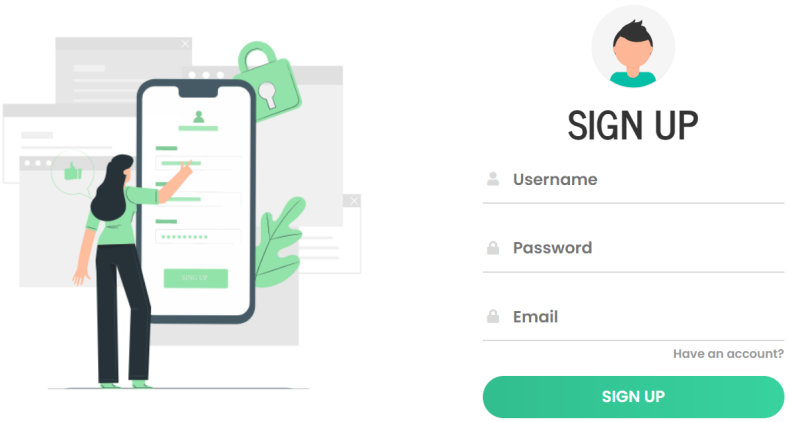


Рисунок 10 - Сторінка реєстрації

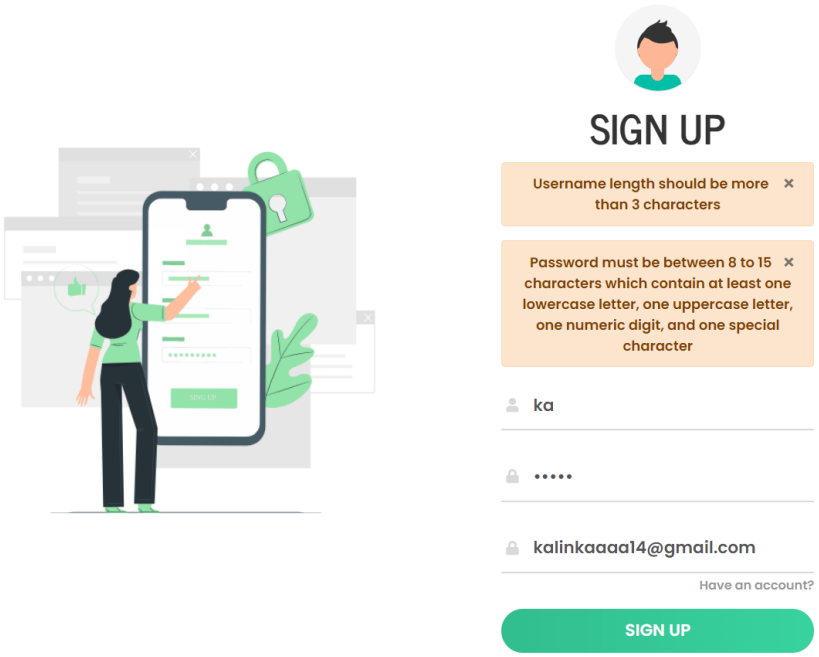


Рисунок 11 - Невдала реєстрація користувача

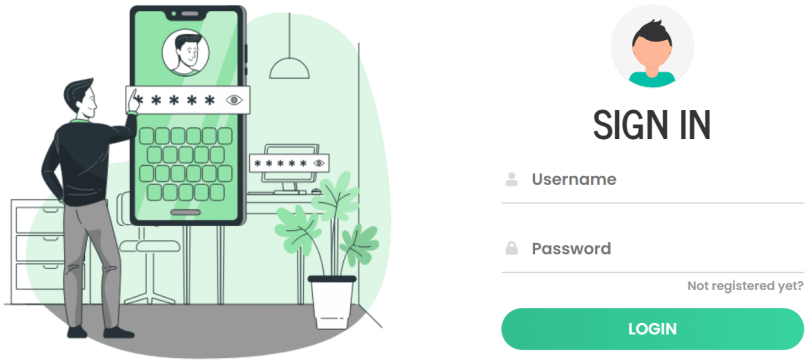


Рис. 12 - Сторінка логіну

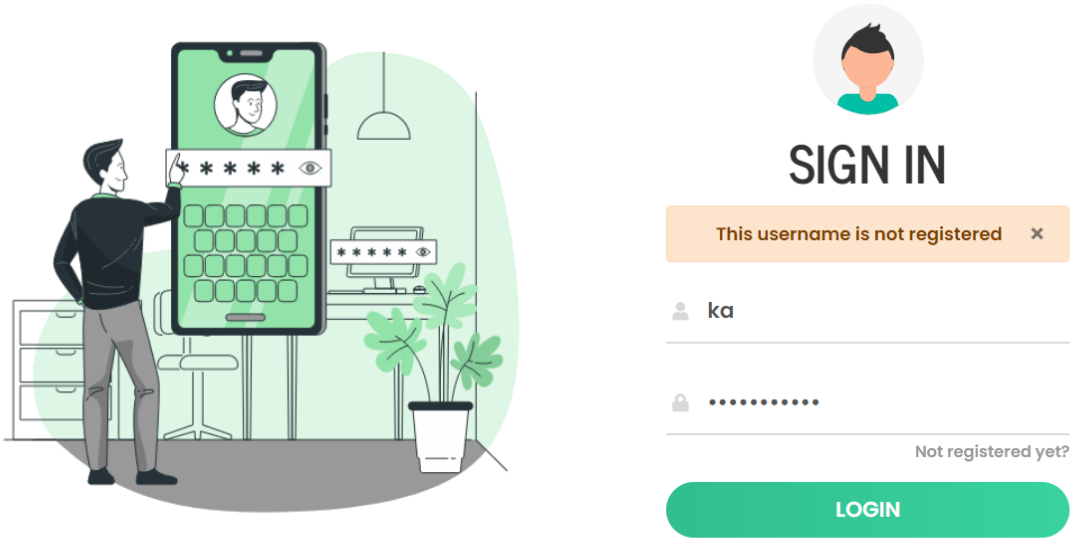


Рис. 13 - Некоректні дані під час автентифікації

У випадку вдалого логінування - користувачу відображується його особистий профіль(рис. 14), на якому він може бачити наступні дані: зверху - навігаційне меню, зліва - перегляд особистих даних, а саме його аватар, який він може за бажанням змінити, натиснувши на кнопку “Choose profile picture”, його

нікнейм, щоденний прогрес, кількість пройдених челенджів та досягнутих звичок з моменту реєстрації. У іншій частині екрану відображаються 3 колонки, де зберігаються всі справи/виклики/челенджі, які призначені на сьогодні.
У випадку, якщо нічого не заплановано - користувачу відображається спеціальне повідомлення про відсутність завдань.

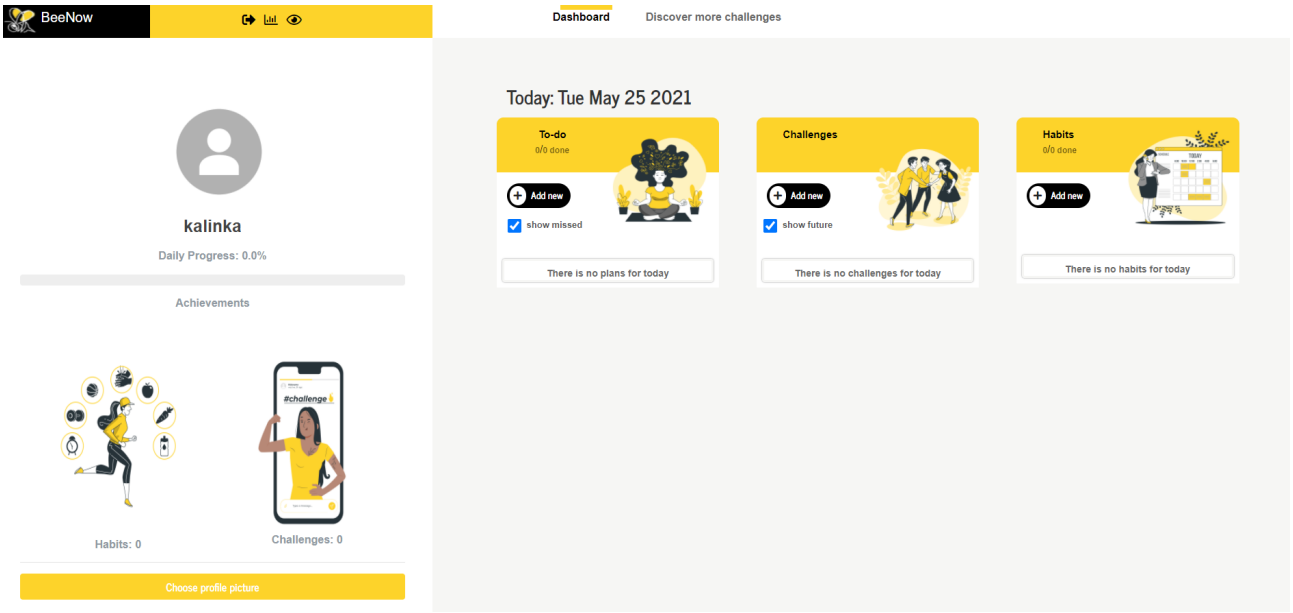


Рисунок 14 - Профіль користувача

Натиснувши на зображення, що знаходяться над обраховними значеннями пройдених звичок та випробувань, юзер може переглянути сторінки з переліком усіх досягнень(рис.15), де буде наявна можливість пройти ще раз даний виклик, або ж засвоїти звичку заново.

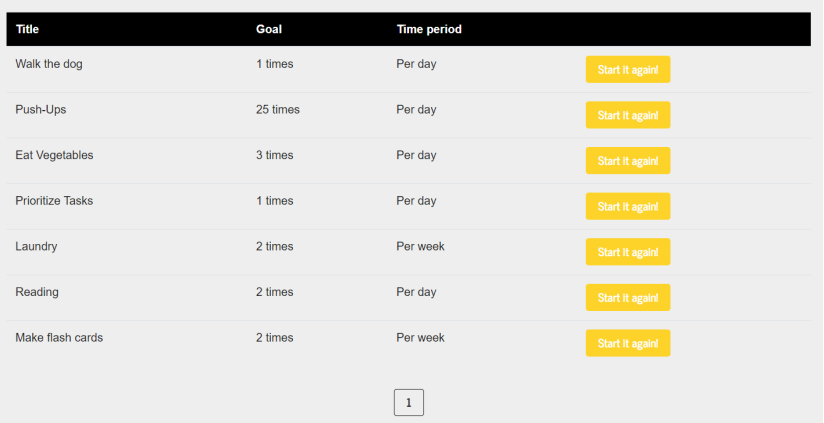


Рисунок 15 - Перегляд досягнень користувача

На сторінці профілю, користувач має можливість додати нову справу, натиснувши кнопку “Add new” у колонці To-do, вказавши назву справи та кінцевий термін виконання(рис. 16).

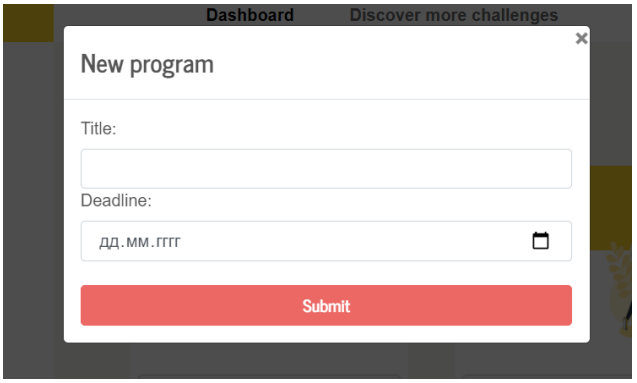


Рисунок 16 - Додавання нової справи

Також, йому надається можливість створити власний челендж, натиснувши на кнопку “Add new” у колонці Challenges та заповнивши всі поля, які є обов’язковими: ціль випробування, яку можна обрати зі списку або ж вписати свою власну, назва, дата старту, тривалість, короткий опис, кроки виконання. Для того, аби додати більше ніж 1 крок, користувачу потрібно натиснути на кнопку “Add new field+”, яке створить нове поле для вводу. У випадку, якщо користувач передумав вводити певну умову, у нього присутня можливість видалити дане поле, натиснувши кнопку “Delete” навпроти обраної умови(рис.13). Крім того, існує поле “Explain why to take this challenge”, яке може бути порожнім, адже є необов'язковим критерієм. За замовчанням, кожне випробування є приватним, тобто є доступним лише тому, хто створює даний челендж та не відображається для інших користувачів у загальному пошуку. Проте під час створення виклику наявна можливість зазначити його публічним. В такому випадку - випробування буде доступним для всіх(рис.17).

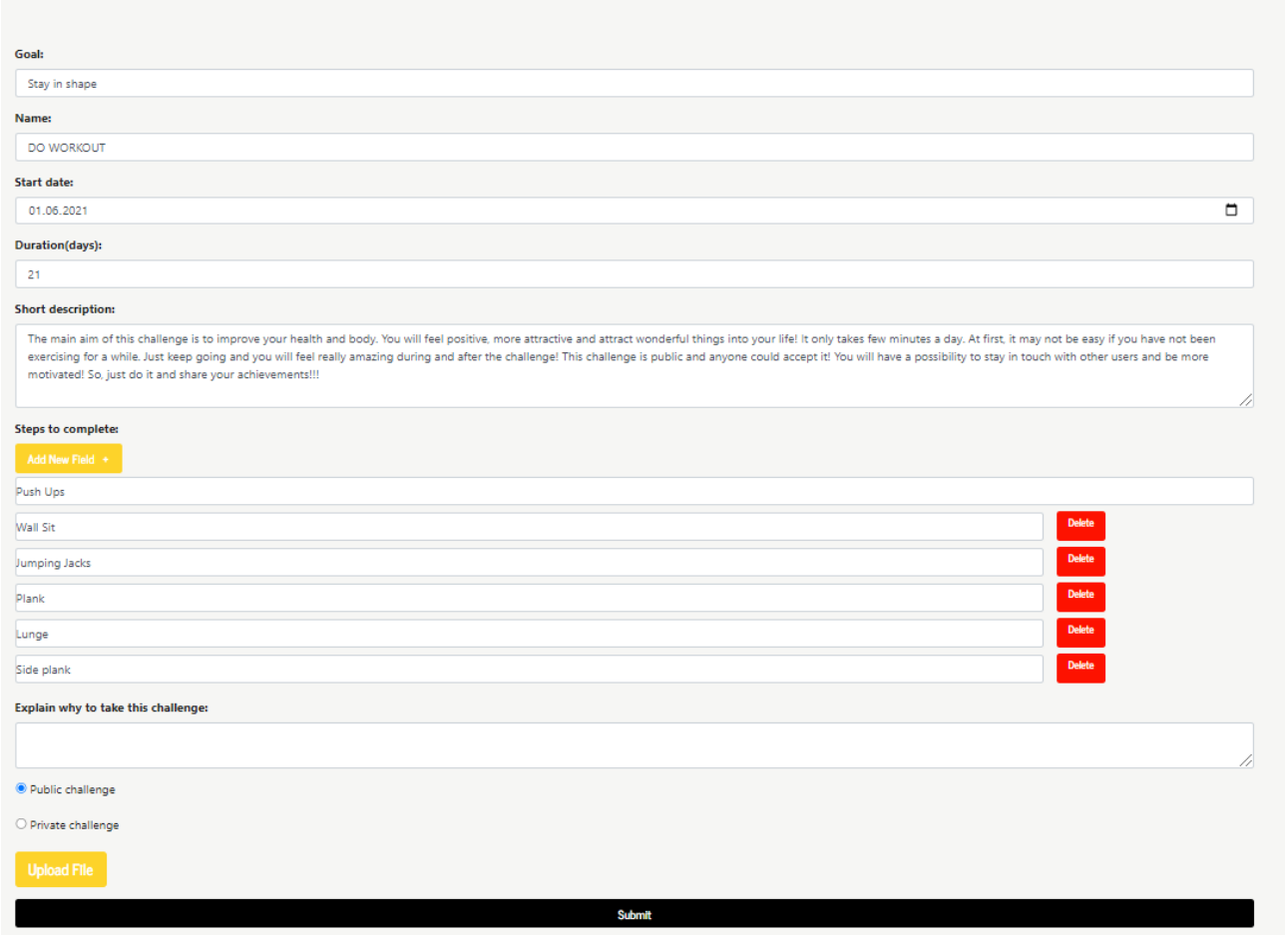


Рисунок 17 - Створення нового челенджу

Після того, коли користувач створив хоча б 1 челендж, його навігаційне меню зміниться так, як це зображено на рис.18.

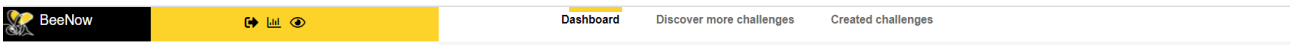


Рисунок 18 - Навігаційне меню користувача, який має власні випробування

Для того, аби створити нову звичку, користувачу потрібно натиснути кнопку “Add new” у колонці Habits на сторінці профілю, яке буде висвічувати модальне вікно для користувача, яке містить 2 способи створення звички - самостійно, або ж за допомогою заготовленого шаблону(рис.19).

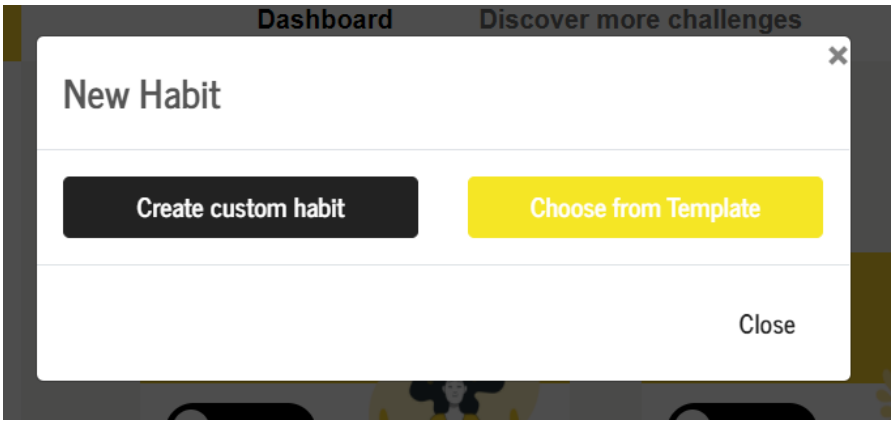


Рисунок 19 - Обрати спосіб створення звички

Якщо обрати 1 варіант(рис.20), користувачу потрібно буде вручну вводити назву звички, дату початку її виконання, її тип, ціль та період часу. У разі виникнення бажання добавити створений екземпляр до заготовлених шаблонів, потрібно обрати “add to templates”, після чого обрати назву шаблону, до якого можна віднести дану звичку, або ж створити свій власний. Після натиснення кнопки “Submit”, у колонці Habits будуть відображатись всі звички, які призначені на сьогодні.

Якщо обрати 2 варіант(рис.21), тобто створення за допомогою заготовлених шаблонів, користувачу буде відображатись список усіх раніше доданих шаблонів. Обравши один елемент з цього списку, у наступному полі буде зберігатись перелік усіх звичок, які належать даному шаблону. Обравши одну із них, усі наступні поля, окрім дати початку виконання, автоматично заповнюються. У разі, якщо певне поля не влаштовує користувача, йому наявна можливість змінити його значення, не змінюючи сам шаблон.

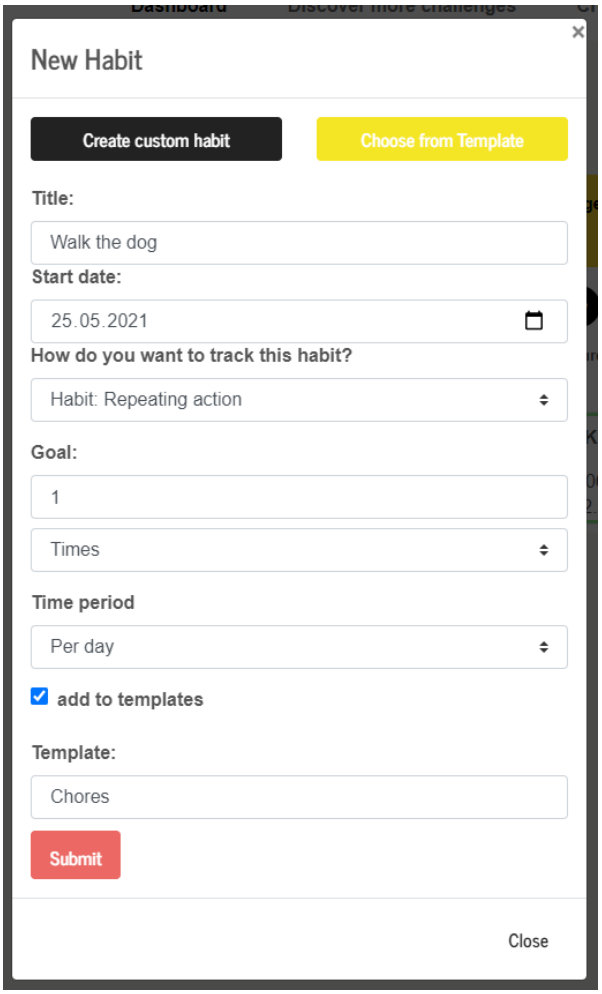


Рисунок 20 - Створення звички власноруч

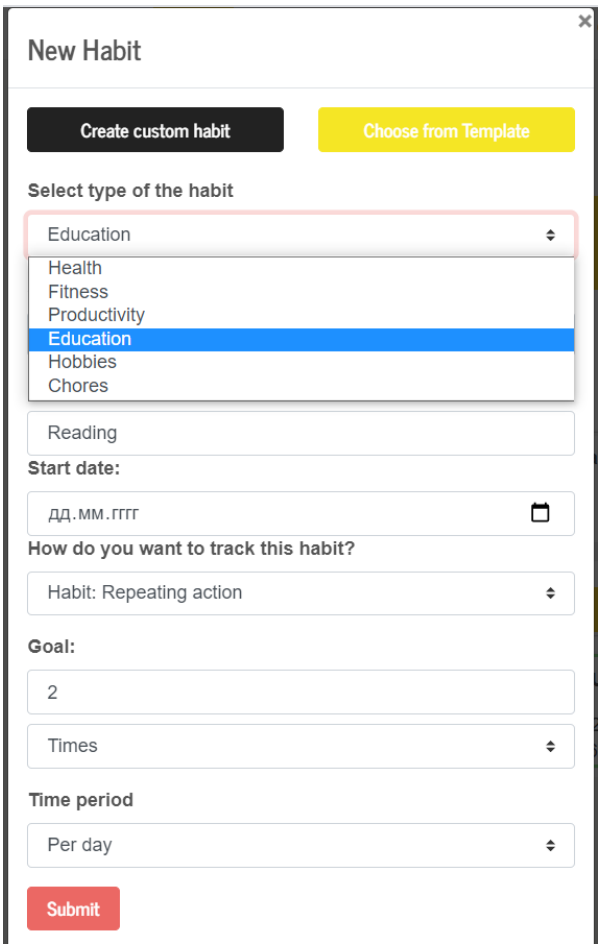


Рисунок 21- Створення звички за допомогою шаблону

Усі раніше створені справи/звички/виклики будуть відображатись у профілі користувача(рис.22). Усі справи є відсортованими за зростанням дати кінцевого терміну здачі, що дозволяє користувачеві бачити всі термінові завдання одразу. За замовчуванням, також показуються пропущені справи із попередніх днів у колонці To-do, які обведені червоною рамкою, та заплановані челенджі у колонці Challenges, які обведені зеленою рамкою. За бажанням, їх можна приховати, вимкнувши відповідно чекбокси show missed/show future. У такому випадку профіль користувача буде виглядати, як на рис.23.

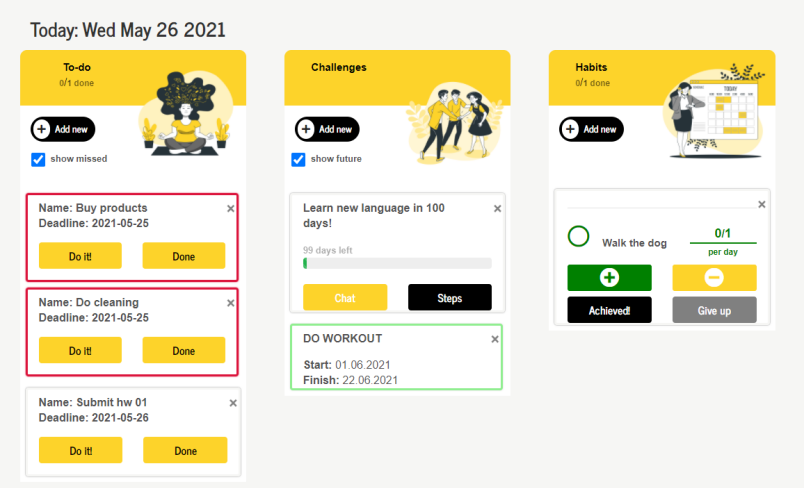


Рисунок 22 - Профіль користувача з пропущеними завданнями та запланованими челенджами

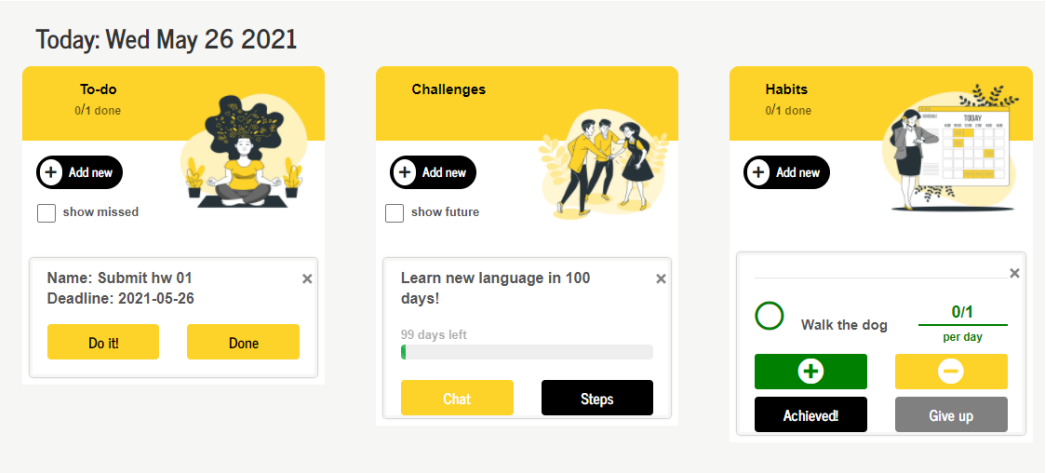


Рисунок 23 - Профіль користувача з прихованими пропущеними завданнями та запланованими челенджами

На профілі користувача у кожній справі з To-do колонки існує 2 можливості: позначити її виконаною, натиснувши кнопку “Done”, тим самим забравши із списку справ, або ж перейти на сторінку з настроюваним таймером нажавши кнопку “Do it”, де використовується техніка Помодоро для кращого фокусування на виконанні справи(рис. 24). У таймері є можливість вказувати кількість хвилин фокусу на справі та відпочинку. Натиснувши кнопку “Start” розпочнеться відлік, який можна зупинити, нажавши “Pause”, відновити, нажавши “Reset” та позначити виконаною за допомогою кнопки “Done”, що

перенаправить на головну сторінку профілю. Коли встановлений час добігає кінця, користувач почує дзвінок таймеру.

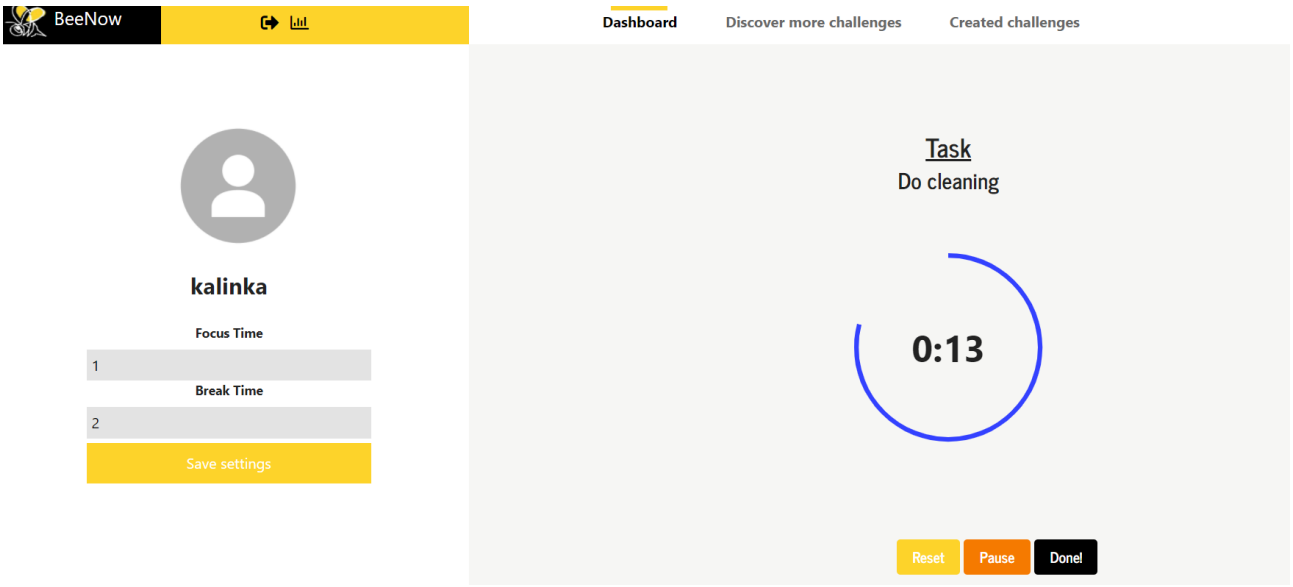


Рисунок 24 - Помодоро Таймер

На профілі користувача у кожному випробуванні, яке є публічним, присутні 2 можливості:

1. Переглянути умови челенджу, позначити кожен пункт виконаним або невиконаним(рис.25).
2. Перейти в чат челенджу, який доступний лише учасникам випробування.

Якщо випробування є приватним для користувача, то можливість чату відсутня.

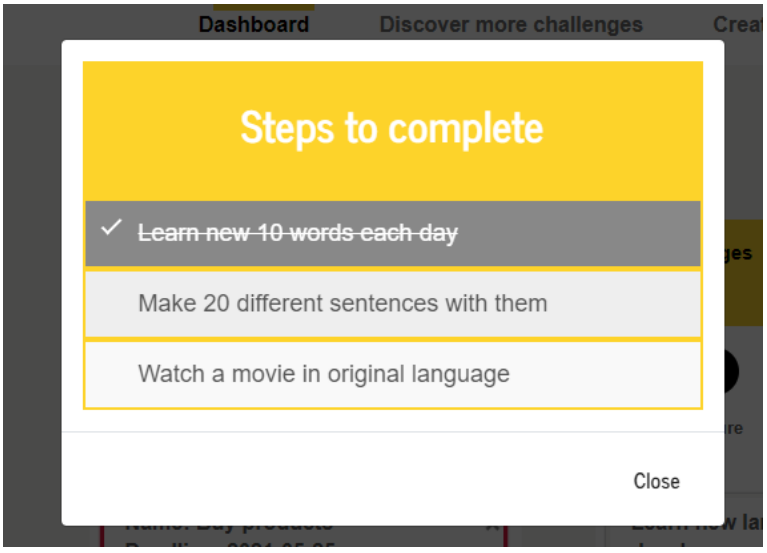


Рисунок 25 - Виконання умов челенджу

Перейшовши в чат випробування, зліва користувачеві будуть відображатись кроки виконання, які можна приховати, нажавши у навігаційному меню кнопку з іконкою ока. У інший частині екрану - вікно, у якому знаходять всі раніше надіслані повідомлення. Так як даний функціонал створений на сокетному зв'язку, усі користувачі, що приєднались до чату, можуть бачити повідомлення інших користувачів миттєво, не перезавантажуючи сторінку ще раз. У випадку, коли виконані всі кроки, з'являється кнопка "Share your achievements" після натиску якої надсилається повідомлення в чат про успіхи користувача(рис.26).

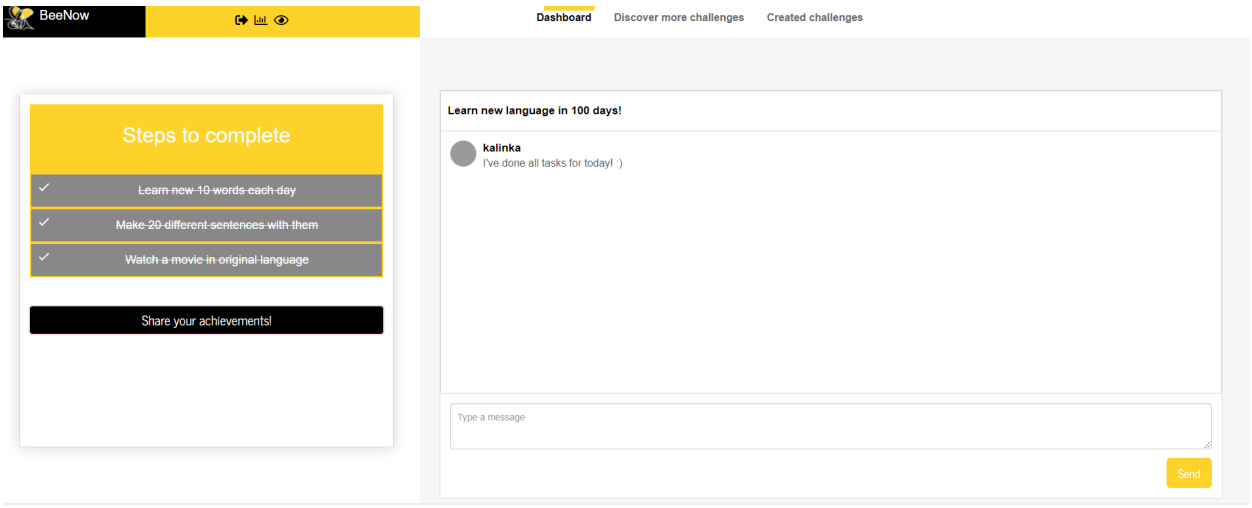


Рисунок 26 - Чат челенджу

У навігаційному меню наявна можливість переходу на сторінку із пошуком та фільтрацію всіх раніше створених публічних челенджів. Фільтрація відбувається за встановленням певної цілі та часових рамок, під час яких триває виклик. За допомогою даних фільтрів, користувач має змогу дізнатись, котрі випробування вже закінчились, котрі тривають, а котрі лише заплановані(рис.27). Кожен челендж з переліку має свою власну сторінку, де описана більш детально інформація про кожен із них. Перейти на неї можна, нажавши кнопку "View details".

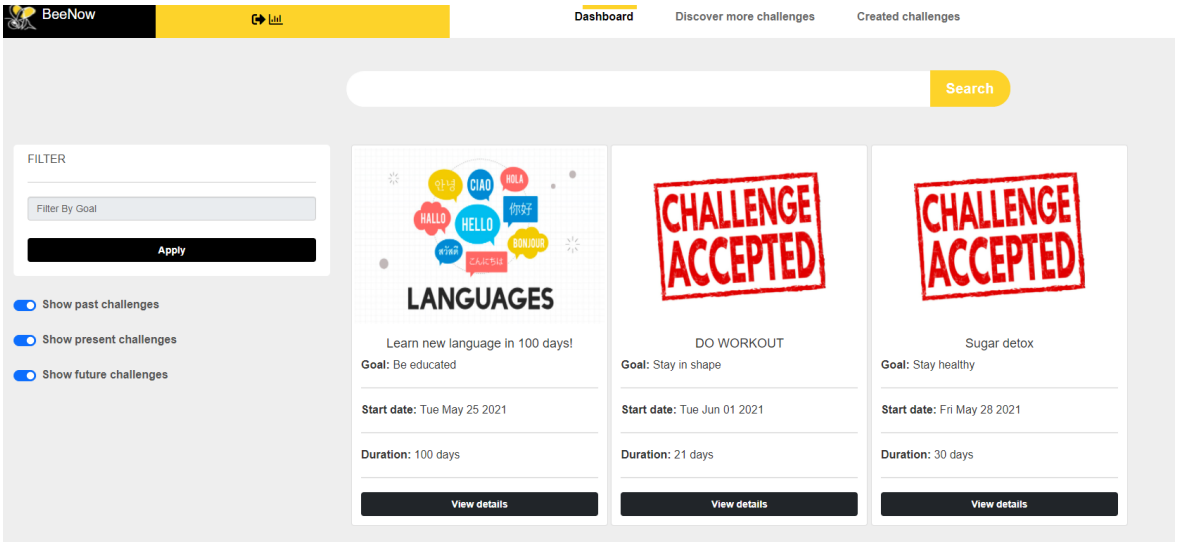


Рисунок 27 - Перелік публічних челенджів

На сторінці челенджу(рис.28) користувач має змогу прочитати всю наведену інформацію, що була на сторінці пошуку всіх випробувань і також детальніше опис виклику та умови, які потрібні для його проходження. Крім того, якщо користувач ще не приєднаний до даного випробування, у нього наявна можливість розпочати його, натиснувши кнопку “Start challenge”.

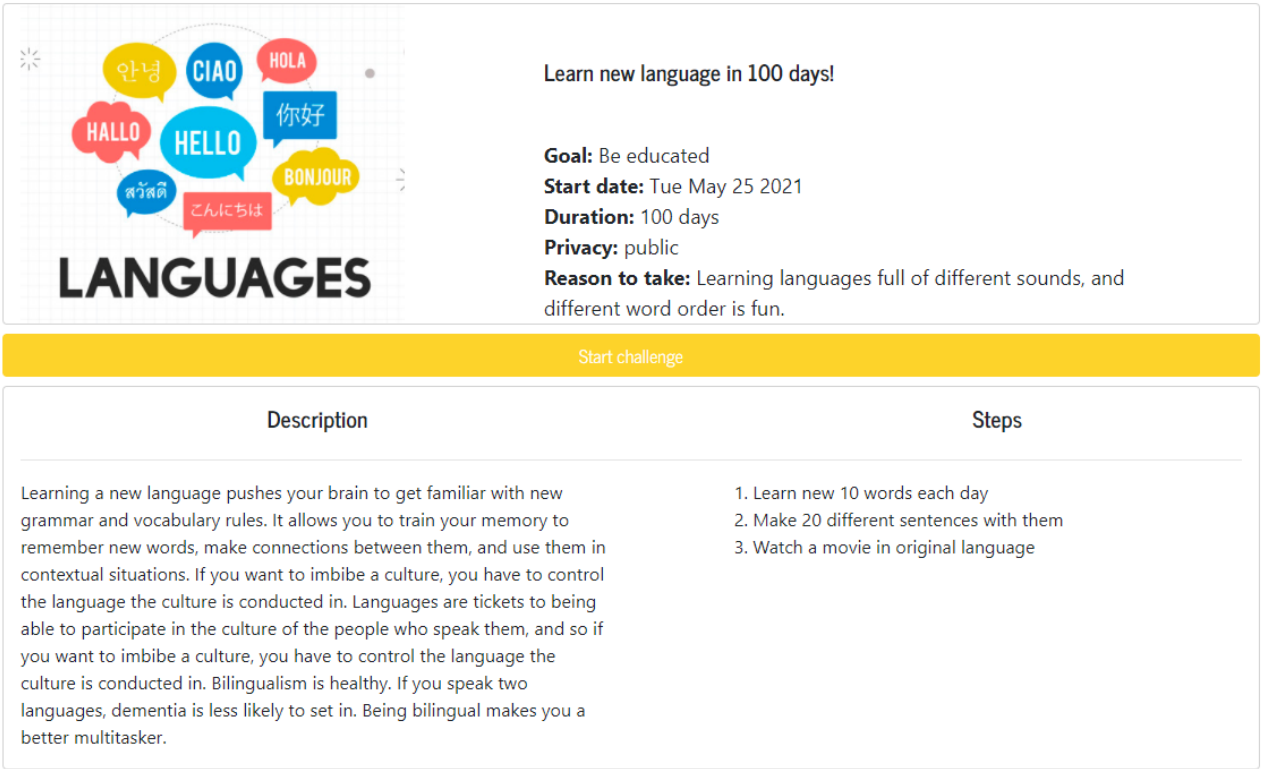


Рисунок 28 - Сторінка челенджу

Якщо користувач бажає приєднатись до завершеного челенджу, то він має змогу проходити його лише приватно, вказавши для себе дату старту. У випадку, якщо випробування триває зараз, або ж заплановане у майбутньому, користувачу надається можливість обрати: приєднатись до наявного виклику і проходити разом з іншими користувачами, чи запланувати собі на окремі дати(рис. 29).

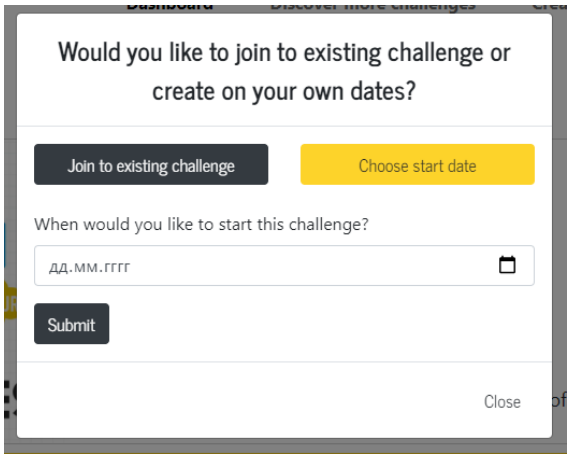


Рисунок 29 - Приєднання до челенджу

Кожен користувач, що створив власні випробування, має змогу перейти по вкладці “Created challenges” (рис.30) у навігаційному меню та переглянути їх. У разі, якщо виклик ще не розпочався, він може редагувати всі його поля, крім назви, цілі випробування та зображення(рис.31) або видаляти, при цьому зміни відбудуться у всіх користувачів, які підписались на виконання даного челенджу.

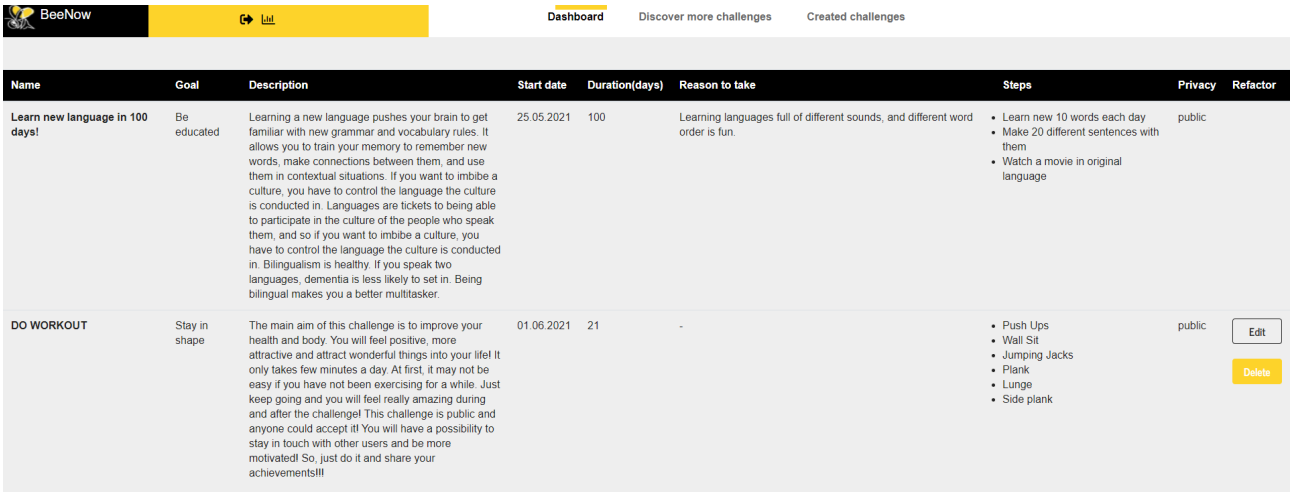


Рисунок 30- Перелік створених челенджів користувачем

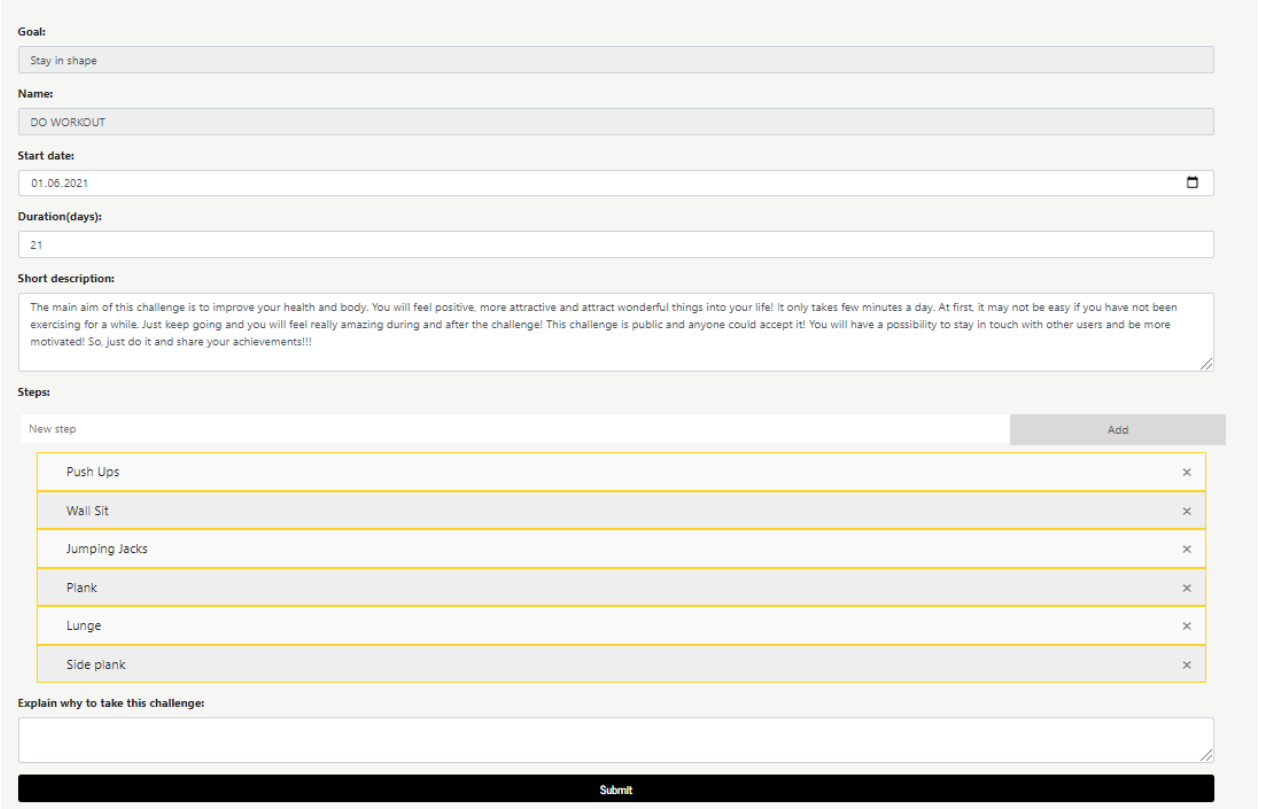


Рисунок 31 - Редагування створеного челенджу

На головній сторінці користувача у колонці Habits зберігаються всі звички користувача на сьогодні. Ті, що ще не досягли своєї цілі за день, поточний тиждень, чи місяць, розташовані вверху даного списку, так як вони пріоритетніші для виконання користувачем. У разі досягнення цілі, звичка переміститься у кінець списку. Кожна створена звичка, в залежності від її типу, може додаватись або відніматись певну кількість раз. Якщо тип звички - “Repeat action”, то обраховне значення виконаних дій буде змінюватись на 1, якщо тип - “Average:Repeat number” , то під час додавання/віднімання користувач має ввести цифру, яку бажає додати/відняти від вже досягнутого нарахованого значення(рис. 32).

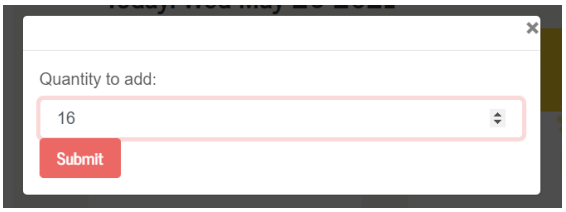


Рисунок 32 - Нарахування виконаної кількості одиниць для звички типу Average

Крім того, у навігаційному меню присутня можливість переглянути статистику за поточний день, місяць, чи останні 3 місяці, що вираховується за щоденним прогресом роботи користувача(рис. 33).

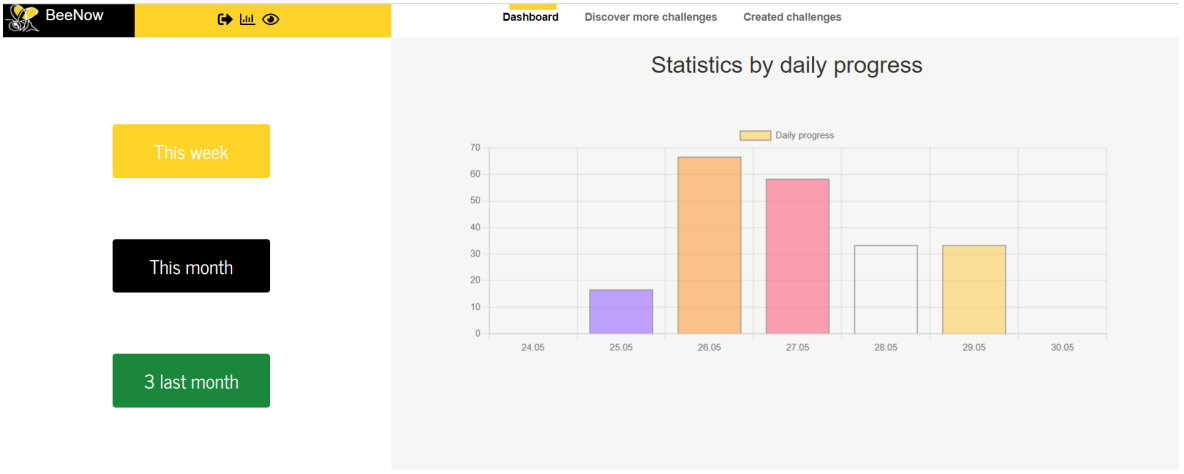


Рисунок 33 - Статистика за поточний тиждень

Якщо залогінився адміністратор сайту, йому доступна лише 1 сторінка(рис.34), на якій відображаються усі запити від користувачів на додавання створених ними звичок у загальні шаблони. Якщо дані коректні та унікальні, адмін має змогу прийняти цей запит, натиснувши кнопку “Accept”, або ж відхилити, натиснувши кнопку “Decline”.

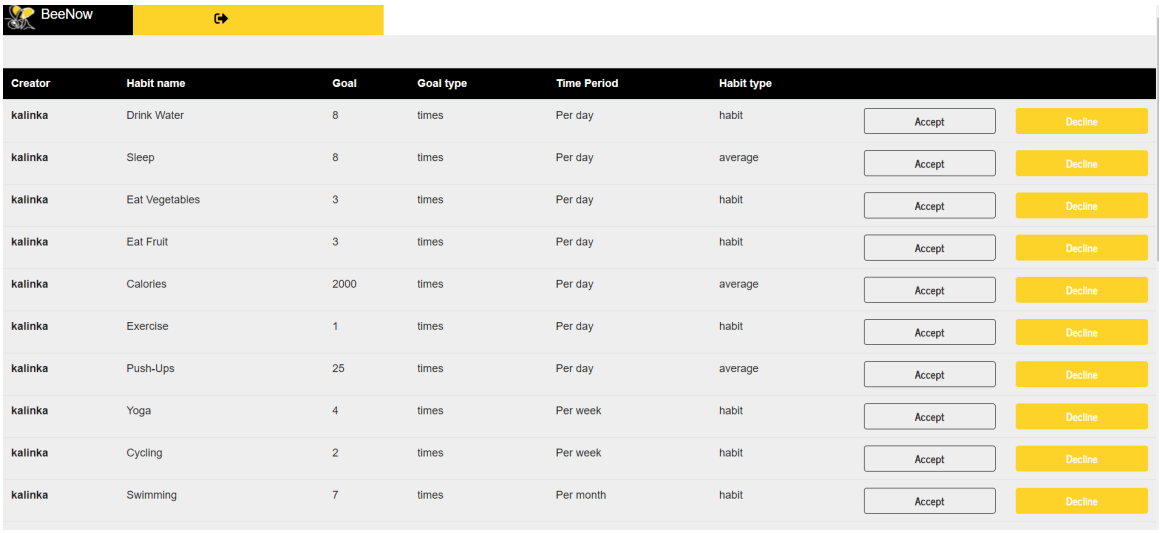


Рисунок 34 - Сторінка адміністратора

3.7 Висновки до третього розділу

Отже, під час виконання третього розділу, було створено повноцінний додаток, який повністю задовольняє вимоги, що зазначались у технічому завданні(п.3.1). Додаток “BeeNow” є багатофункціональним, який включає в себе не тільки можливості вироблення нових звичок та відслідковування власного прогресу, але й платформою для комунікації та спілкування з іншими користувачами. Усі технології, що були описані у розділі 3.2 були використані під час написання застосування. Після створеного додатку було здійснене мануальне тестування, у якому перевірялись коректність створених функцій розробником і яке пройшло успішно.

Висновки по роботі та аналіз можливостей подальшого розвитку застосунку

Отже, під час даної курсової роботи було виконане веб-застосування, яке є актуальним у XXI столітті, та може допомогти будь-якому користувачеві мережі Інтернет покращити навики тайм-менеджменту та навчитись правильно для себе виставляти пріоритети, таким чином не просто встигати все за день, але й мати вільний час для себе. Даний додаток може замінити навіть кілька інших застосувань водночас, адже він має досить багато функцій, які будуть корисними для будь-якого користувача.

Незважаючи на те, що усі функції у додатку працюють коректно, наявні наступні ідеї для покращення створеного додатку та подальшого розвитку:

1. Варто оптимізувати якість зображень, аби уникнути занадто розтягнутих, чи зжатих.
2. Можливість відправляти в чат зображення, емоджі, голосові повідомлення.
3. Можливість добавлення аудіо-,відео-завдань до челенджу.
4. Підключення сторонніх ресурсів, таких як Google Fit та Google Calendar, для синхронізації даних з веб-трекером.
5. Створення можливості створювати челендж за готовим шаблоном.
6. Зробити можливість оцінювання пройденого челенджу.
7. Додати можливість перегляду календаря з усіма минулими і майбутніми справами.
8. Створити вкладку сповіщень, де будуть оновлювати інформація, якщо користувач завершив челендж.
9. У випадку, коли адміністратор оновлює челендж, зробити нове сповіщення юзеру з можливістю відписатись від зміненого челенджу.

Список використаної літератури

1. Стаття про опис роботи клієнт-серверної архітектури [Електронний ресурс] https://www.w3schools.in/what-is-client-server-architecture/
2. Стаття про пояснення роботи протоколу HTTP [Електронний ресурс] https://javarush.ru/groups/posts/2521-chastjh-3-protokolih-httphttps
3. Відео-пояснення можливостей Chart.js [Електронний ресурс] https://www.youtube.com/watch?v=sE08f4iuOhA
4. Стаття з переліком класифікації веб-сайтів 1 частина [Електронний ресурс] https://www.gwsmedia.com/20-different-types-websites-1
5. Стаття з переліком класифікації веб-сайтів 2 частина [Електронний ресурс] https://www.gwsmedia.com/articles/20-different-types-websites-part-2
6. Стаття про пояснення роботи AJAX [Електронний ресурс] https://www.tutorialspoint.com/jquery/jquery-ajax.htm
7. Стаття про призначення мови програмування JavaScript [Електронний ресурс] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript
8. Посилання на сторінку бібліотеки Chart.js для побудови графіків [Електронний ресурс] https://www.chartjs.org/
9. Стаття про переваги використання нереляційної бази даних Монго [Електронний ресурс] https://www.tutorialspoint.com/mongodb/mongodb_advantages.htm
10. Стаття із порівнянням різних додатків-аналогів веб-трекеру [Електронниц ресурс] https://collegeinfogeek.com/habit-tracker/

Додатки

1. Схема архітектури застосунку