# РЕФЕРАТ

Кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю інженерія програмного забезпечення. – Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм-61, м. Тернопіль, 2021. сторінок – , рис. – 42, табл. – 2, додатків – 4, бібліогр. – 20.

Тема кваліфікаційної роботи магістра присвячена розробці веб-орієнтованої системи планування витрат у сфері будівництва.

Об’єктом дослідження є застосування інформаційних технологій у сфері будівництва.

Метою кваліфікаційної роботи магістра є здійснення аналізу предметної області, проектування архітектури та розробка веб-орієнтованої системи планування витрат у сфері будівництва на основі JavaScript технологій.

В роботі здійснено аналіз предметної області, а саме огляд використання інформаційних технологій при складанні та веденні плану витрат на проведення будівних робіт. Визначено функціональні вимоги програмного продукту та проаналізовано обрані веб-технології реалізації програмного забезпечення. У роботі здійснено проектування архітектури системи, моделювання варіантів використання, діаграми діяльності, розробки та проектування моделі бази даних.

Результатом виконаної кваліфікаційної роботи магістра є реалізована веб-орієнтована система планування витрат у сфері будівництва, практичне значення якої полягає у обчисленні енергоефективності будинку і плануванні заходів щодо її підвищення та покращення будинку; можливості створення та ведення плану проекту із оцінкою вартості кожної з категорій витрат.

Ключові слова: ЕНЕРГОЕФЕКТИВНІСТЬ, ПЛАНУВАННЯ ВИТРАТ, ПЛАН, ПРОЕКТ, СЕРВЕР, REACT, EXPRESS, JAVASCRIPT, MONGODB.

# ABSTRACT

Qualification work for a master's degree in software engineering. - Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, SPm-61 Group, Ternopil, 2021. pages – , fig. – 42, tables – 2, appendices – 4, bibliogr. – 20.

The topic of the master's qualification work is devoted to the development of a web-based cost planning system in the field of construction.

The object of research is the use of information technology in the field of construction.

The purpose of the master's qualification work is to analyze the subject area, design architecture and develop a web-based cost planning system in the field of construction based on JavaScript technology.

The analysis of the subject area is carried out in the work, namely the review of use of information technologies at drawing up and conducting the plan of expenses for carrying out construction works. The functional requirements of the software product are determined and the selected web technologies of software implementation are analyzed. The system architecture design, usage modeling, activity diagrams, database model development and design are performed in the work.

The result of the master's qualification work is a web-based system of cost planning in the field of construction, the practical significance of which is to calculate the energy efficiency of the house and plan measures to increase and improve the house; opportunities to create and maintain a project plan with an estimate of the cost of each of the cost categories.

Keywords: ENERGY EFFICIENCY, COST PLANNING, PLAN, PROJECT, SERVER, REACT, EXPRESS, JAVASCRIPT, MONGODB.

**ЗМІСТ**

# ВСТУП

Актуальність роботи полягає в тому, що питання про енергоефективність було і залишається одним із пріоритетних напрямів економіки, тому, здійснивши аналіз даної проблеми, виникає необхідність проектування та розробки програмного забезпечення для обчислення енергоефективності будинку, а також процесу планування будівельних робіт щодо підвищення її рівня. Актуальність енергозбереження та підвищення енергетичної ефективності будівель обумовлено високими витратами та постійним зростанням тарифів на енергоресурси. Цілеспрямована реалізація програм енерго-ресурсозбереження дозволила б при значно менших, ніж для введення нових енергетичних потужностей, капітальних витрат, знизити дефіцит енергії та створити сприятливі умови для вирішення проблем у паливно-енергетичному комплексі.

Планування будівництва є фундаментальним і складним видом діяльності в управлінні та виконанні будівельних проектів. Це передбачає вибір технології, визначення робочих завдань, оцінку необхідних ресурсів і тривалості виконання окремих завдань, а також виявлення будь-яких взаємодій між різними робочими завданнями. Хороший план будівництва є основою для планування бюджету та графіку робіт. Розробка плану будівництва є важливим завданням в управлінні будівництвом, навіть якщо план не написаний або іншим чином не зафіксований. На додаток до цих технічних аспектів планування будівництва також може знадобитися приймати організаційні рішення про відносини між учасниками проекту і навіть те, які організації включити в проект.

На сьогоднішній день розроблена велику кількість програмного забезпечення для розрахунку витрат у сфері будівництва. Проте більшість з досліджених аналогів передбачають платну підписку та мають функціонал розрахований на будівельні компанії зі складною системою обчислень. Це ускладнює та обчислення та планування проекту для звичайних користувачів. Таким чином, актуальною є

задача розробки зручного та гнучкого програмного забезпечення для формування плану будівельних робіт, що буде задовольняти всі потреби користувача.

Об’єктом дослідження є застосування інформаційних технологій у сфері будівництва.

Предметом дослідження є програмне забезпечення для планування проекту будівельних робіт, які проводяться з метою підвищення енергоефективності будинку.

Метою кваліфікаційної роботи магістра є проектування та розробка веб-орієнтованої системи планування витрат у сфері будівництва.

Для досягнення мети роботи необхідно виконати такі задачі:

– здійснити аналіз проблеми використання програмного забезпечення при складанні плану енергоефективних будівельних робіт;

– здійснити аналіз існуючих поширених програмних рішень, які використовуються при плануванні витрат на проведення будівельних робіт, скласти вимоги до програмного продукту;

– провести дослідження популярних технологій розробки веб систем для планування процесу ведення проекту з будівельних робіт;

– розробити веб-орієнтовану систему складання плану будівельних робіт та провести її модульне та інтеграційне тестування.

Дотримуючись плану поставлених задач, на останньому завершальному кроці, буде спроектовано та реалізовано програмний продукт, який надасть можливість розрахувати кількість викидів вуглекислого газу, який виробляється житловими будинками, зручно та швидко скласти гнучкий план проекту по проведенню будівельних робіт з обліком всіх здійснених платежів, по кожній із планованих категорій витрат.

# 1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

## 1.1 Аналіз вимог до програмного забезпечення

### 1.1.1 Аналіз предметної області

На сьогоднішній день у Європі будівлі значну кількість викидів парникових газів через те, що вони використовують 40% первинної енергії. Близько 85% енергії, споживаної в будівлях, використовується для опалення, освітлення та нагрівання води для повсякденних потреб. Велика частина цієї енергії може бути зекономлена за рахунок збільшення теплоізоляції будівлі і використання ефективних систем опалення або охолодження. Рішенням для стійких будівель є інтеграція систем відновлюваних джерел енергії, таких як сонячні панелі для виробництва електроенергії та сонячні теплові панелі для нагріву побутової води. Європейський союз затвердив, що споживання енергії в будівлях має бути скорочено на 20%, а викиди парникових газів – до 18% [1]. Пасивний будинок перетворився на бажану концепцію для архітекторів і дослідників у багатьох країнах (наприклад, у Німеччині, Швейцарії, Австрії), оскільки він продемонстрував високий тепловий комфорт і низьке енергоспоживання.

Пасивний будинок – це будівля, яка має комфортну температуру в приміщенні взимку або влітку, з низьким енергоспоживанням для опалення або охолодження приміщення [2]. Пасивні будинку є екологічно чистими за визначенням: вони використовують надзвичайно мало первинної енергії, залишаючи достатньо енергетичних ресурсів для всіх майбутніх поколінь, не завдаючи жодної шкоди навколишньому середовищу. Додаткова енергія, необхідна для їх будівництва є досить незначною в порівнянні з енергією, яку вони заощаджують пізніше. Варто зазначити, що стандарт пасивного будинку забезпечує такий рівень стійкості для всіх, хто бажає побудувати нове будівництво або відремонтувати старе за доступною ціною – внесок у захист навколишнього середовища. Пасивні будинки є як і доступні так і заощаджують кошти в тривалій перспективі. Інвестиції в будівельні компоненти вищої якості, які вимагаються

стандартом пасивного будинку, пом’якшуються виключенням дорогих систем опалення та охолодження. Додаткова фінансова підтримка, яка все частіше доступна в багатьох країнах, робить будівництво пасивного будинку ще більш можливим. На рисунку 1.1 зображено ключові особливості проектування пасивного будинку.

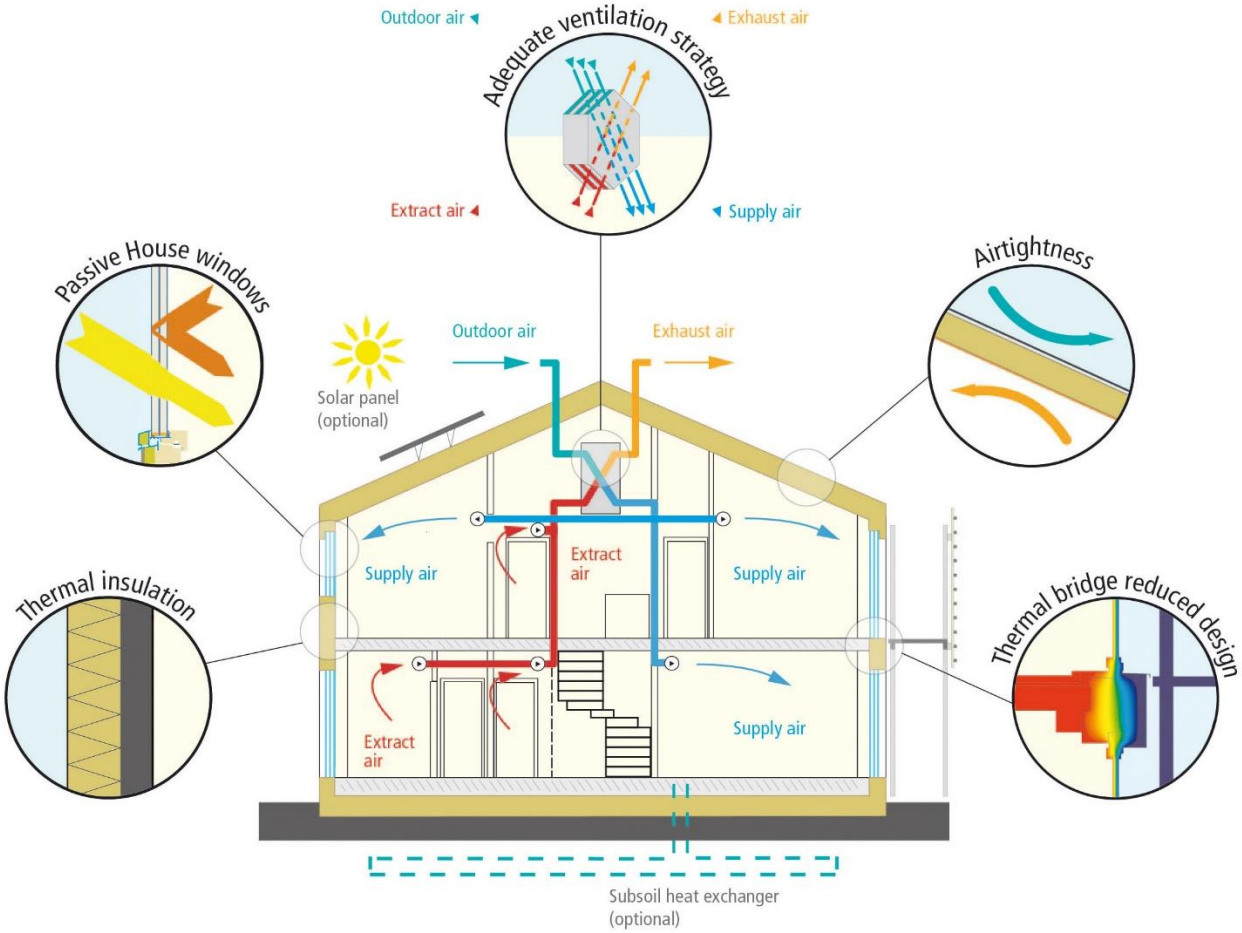



Рисунок 1.1 – Ключові особливості проектування пасивного будинку

Концепція пасивного будинку була вперше використана в Німеччині, і відноситься до будинку з наступними характеристиками [3]:

1. Теплоізоляція. Всі непрозорі будівельні компоненти зовнішньої сторони будинку повинні бути добре ізольовані. Для країн з холодним кліматом це означає, що коефіцієнт теплопередачі не перевищує 0,15Вт/(м2К), тобто втрачає максимум 0,15Вт на градусі різницю температури і на квадратний метр зовнішньої поверхні.

2. Вікна пасивного будинку. Віконні рами повинні бути добре ізольовані та оснащені низькоенергетичними склопакетами, заповненими аргоном або криптоном для запобігання теплопередачі. Для країн з холодним кліматом це означає значення U 0, 80 Вт/(м2К) або менше, при значенні загального коефіцієнту пропуску соняної енернії g близько 50 %.

3. Вентиляція з рекуперацією тепла. Ефективна вентиляція з рекуперацією тепла є ключовим фактором, що забезпечує хорошу якість повітря в приміщенні та економіку енергії. У пасивному будинку за допомогою теплообмінника близько 70% тепла від відпрацьованого повітря знову передається свіжому повітрю.

4. Герметичність будівлі. Неконтрольований витік через зазори має бути менше 0,6 від загального обсягу будинку на рік під час вимірювання при тиску 50 Паскаль.

Концепція пасивного будинку заснована на зниженні теплових втрат. Це досягається за рахунок високої теплоізоляції і підвищеної теплоємності оболонки, що дозволило б зберігайте енергію всередині; пасивного приросту сонячної енергії; ефективної система рекуперації тепла. У світі існують різні стійкі будівлі, що мають стандартний дизайн, які використовуються для оцінки енергоефективності будівель, таких як BREEAM (Великобританія), LEED (США), Passive House (Німеччина). BREEAM означає Метод оцінки навколишнього середовища дослідницької установи в галузі будівництва і являє собою провідний в світі метод оцінки стійкості для проектів генерального планування, інфраструктури та будівель. BREEAM представляє значення вимірювань всіх категорій в будівлях від енергетики до екології.

Розробники можуть впроваджувати відносно прості підвищення ефективності в будинках, щоб зменшити споживання енергії та витрат. Проекти охоплюють нове будівництво або реконструкцію існуючих будинків шляхом встановлення енергоефективних та відновлюваних джерел енергії для опалення, охолодження та освітлення; покращена теплоізоляція та ефективність використання води; і використання екологічно чистих будівельних матеріалів,

таких як сертифікована деревина. Фінансування проектів сталого розвитку в житловому секторі є перспективним ринком для фінансових установ, оскільки попит на житло та енергію продовжує зростати.

Зростання попиту на доступне житло та енергію: понад два мільярди людей будуть додані до зростаючого глобального попиту на житло, енергію та міську інфраструктуру протягом наступних 25 років. Попит на енергію в житловому секторі може становити до 50% загального споживання енергії в країні, що збільшує витрати домогосподарств у багатьох країнах [4]. Великий потенціал енергоефективності житла: підвищення енергоефективності стає все більш економічним і має великий потенціал для застосування в існуючих житлових будинках та індивідуальних сімейних будинках. У деяких країнах, що розвиваються, діяльність також зосереджена на будівництві стійкого житла.

Будинок працює як система, усі його елементи – система вентиляції, опалення та охолодження, зовнішнє середовище і навіть діяльність мешканців – впливають один на одного. Від того, як ці елементи взаємодіють, залежить загальна продуктивність будинку. Наприклад, погана ізоляція або вентиляція може перекреслити інвестиції в нову систему опалення або оновлені вікна та двері. Підвищення енергоефективності по-справжньому окупляться, лише якщо звернутись до всієї системи. Житло не повинне бути новим, щоб бути енергоефективним. Ми можете модернізувати свій будинок і отримати результати з мінімальними витратами.

Змінення підходу до вибору та використання техніки. Купівля енергозберігаючих приладів – хороший початок. Клас енергоефективності, показники енергоспоживання, які вказуються на приладах, зокрема, на холодильниках, пральних машинах та інших приладах, дають знати, наскільки енергоефективним є прилад. Можна шукати прилади з високим рейтингом і дізнатися, скільки можна заощадите на рахунках за електроенергію.

Встановлення сонячних фотоелектричних панелей. Понад два мільйони австралійських будинків тепер мають сонячні панелі на дахах. Оскільки панелі стають дешевшими, все більше австралійців обирають їх встановлення та

зменшують свої рахунки та викиди вуглецю. Будинки з оцінкою восьми зірок і вище часто можуть виробляти всю власну енергію від стандартної сонячної батареї потужністю шість кіловат протягом року.

Ми витрачаємо багато енергії на охолодження повітря влітку та на зігрівання взимку. Тому ми можемо легко встановити віконні та дверні ущільнювачі в більшості старих будинків самостійно. Ми також можемо встановити заслонки димоходу, замінити витяжні вентилятори на вентилятори, що закриваються, закрити старі стінні вентиляційні отвори.

Ми втрачаємо 10-35 відсотків тепла або втрат через вікна та двері з одинарним склопакетом. Дооснащення будинку вікнами та дверима з подвійним або потрійним склопакетом зробить будинок набагато комфортнішим в екстремальну погоду. Це також заощадить кошти та зробить будинок тихіше, оскільки вони блокують зовнішній шум.

Необхідно встановлювати нову систему гарячої води якомога ближче до кранів. Необхідно зробити спеціальну ізоляцію труб з особливою товщиною відставання (25 мм добре). Велика кількість енергії витрачається на транспортування гарячої води в неізольованих мідних трубах. Потрібно використовувати залишки ізоляційних брусків навколо ванни. Це збереже ванну гарячою довше [5].

Якщо ми оновлюємо електропроводку, потрібно переконатися, що всі нові світильники використовують світлодіодні лампи. Якщо побутова техніка потребує заміни, потрібно розглянути вибір приладів з високим рейтингом. Початкові витрати на проект пасивного будинку в середньому на 10-30% вище, ніж при звичайному будівництві.

До рішення про будівництво проекту пасивного будинку можна підійти з фінансової точки зору. Можлива економія енергії понад 90%, але відповідність стандарту пасивного будинку може мати дуже високу вартість в місцях з екстремальними температурами. Власники повинні зважити заощадження на комунальних послугах та витрати на будівництво та вирішити, чи є віддача від інвестицій привабливою.

Отже, пасивний будинок являється хорошим вкладенням, оскільки він не шкодить довкіллю, і зробить величезну економію на елекроенергії та опаленні в майбутньому. Значно знижує витрати на опалення та охолодження в будівлях, і ця концепція не обмежується житловим сектором.

### 1.1.2 Постановка задачі

Веб-орієнтована система планування витрат у сфері будівництва повинна забезпечувати виконання таких функцій:

- здійснення гостьового входу;
- реєстрація в системі за допомогою сервісів: Google, Facebook;
- можливість підключення OTP для авторизації;
- виконання оцінки енергоефективності будинку;
- обчислення кількості викидів $CO_2$ до планування та потенційно можливої кількості після проведення ремонтних робіт;
- обчислення кількості спожитої електроенергії до та після виконання робіт щодо збільшення енергоефективності будинку;
- створення проекту та плану;
- розрахунок фактичних витрат на ремонт житла, можливого потенціалу фінансової економії;
- редагування проекту;
- отримання приблизної оцінки вартості ремонту або купівлі власного житла за введеними користувацькими параметрами;
- можливість отримання особистого консультування з надсиланням запиту для отримання кредиту;
- зміна мови інтерфейсу;
- експортування проектних розрахунків у форматі PDF;

- можливість вибору типу обладнання;
- можливість заповнення форм із параметрами, які описують проект;
- коригування розрахованого плану;
- генерування діаграми по планованих категоріях витрат;
- ведення обліку витрат по кожній із запланованих категорій;
- можливість створення до 10 проектів.

### 1.1.3 Пошук акторів та варіантів використання

Діаграми UML для опису варіантів використання можуть бути забезпечені письмовими описами та ілюстративними малюнками. Не кожна діаграма повинна використовуватися в кожному конкретному випадку. Який тип діаграми слід використовувати, залежить від того, які характеристики системи необхідно підкреслити. У будь-якому випадку рекомендовано створювати діаграми варіантів використання, оскільки цей тип діаграм добре підходить для спілкування з системними партнерами та експертами з предметної області про основні функції та контекст системи.

Діаграма варіантів використання демонструє різні способи взаємодії користувача з системою. Даний вид UML діаграми показує учасників, варіанти використання та їх взаємозв'язок. Ці діаграми дають хороший огляд функціональності системи

Діаграми варіантів використання показують варіанти використання, учасників і зв’язки між ними. Зв’язки між акторами та варіантами використання вказують на те, що актор може використовувати певну функціональність системи.

Діаграма варіантів використання включає такі елементи:

– актант: представляє роль, яку бере на себе стороння людина при взаємодії з системою. Актором може бути клієнт, діловий партнер, постачальник або інша система;

– варіант використання: описує взаємодію між актантом і системою, тобто він описує функціональність системи, яку використовує суб'єкт;

– subject: описує систему, до якої прикріплений один або кілька варіантів використання. Даний елемент представлено прямокутником, який оточує визначені прецеденти.

Діаграма варіантів використання визначає такі типів зв’язків:

– включення – зв'язок між двома варіантами використання, який означає, що один варіант використання, включений в інший;

– асоціація – зв'язок між актантом і прецедентом. Вказує на те, що актант може використовувати певну функціональність системи - варіант використання;

– розширення – зв’язок ідея якого полягає в тому, щоб створити розширюваний або доповнений прецедент, а всередині нього описати, де і за яких умов він розширює поведінку деякого базового прецеденту;

– узагальнення – зв’язок, який необхідно використовувати, коли знайдено два або більше прецеденти, які мають спільні риси призначення та структури. Якщо такі знайдено, необхідно описати їхні спільні частини в новому прецеденті.

При використанні діаграм варіантів використання для моделювання систем і процесів рекомендується підтримувати низький рівень абстракції. Для зрозумілості діаграм і для зв'язку між зацікавленими сторонами краще додати надмірність, ніж занадто багато абстрагуватися.

Принципово важливо, щоб використовувалася термінологія процесів або організації, і щоб описи варіантів використання вибиралися таким чином, щоб їх можна було зрозуміти інтуїтивно.

Термінологія з області інформаційних технологій не відноситься до діаграм варіантів використання на рівні процесів. Змішання термінів з процесу і IT-

спільнот призводить до поганих результатів. Насправді IT-спеціалісти часто стикаються з варіантами використання, які вже дуже близькі до IT на рівні процесів. Це призводить до плутанини в двох аспектах: користувачі - тобто люди, які беруть участь у процесах і не знайомі з IT-термінологією.

Веб-орієнтована система повинна бути доступною та зрозумілою для користувача, який не є спеціалістом у сфері будівництва.  вимог. Планується, що система буде доступна у вільному доступі.

В розроблювальній системі планування витрат для сфери будівництва присутні два актанти: головним є зареєстрований користувач, якому будуть доступні всі функції системи, які відповідають прецедентам; другим актантом є гість, якому доступна лише частина функцій системи. Зі сторони проектування багатокористувацьких можливостей та ролей - система не містить складні архітектурні рішення, оскільки відсутнє планування взаємодії між користувачами.

### 1.1.4 Опис ключових варіантів використання

Дана веб-орієнтована система містить два актори: зареєстрований користувач та гість. Під час першого входу в систему та після вибору планованого типу проекту користувач отримує роль гостя. Гість - користувач системи, має доступ до створення проекту з можливістю подальшої реєстрації в системі, або входу в існуючий аккаунт, але не має доступу до деяких інструментів системи. Зареєстрований користувач - користувач системи, який виконав кроки реєстрації і авторизації, отже відповідно до цього має повний доступ до всіх функцій системи.

Спільними варіантами використання для гостя та зареєстрованого користувача є такі: обчислення енергоефективності будинку, створення проекту, додавання та видалення категорій витрат, редагування інформації про проект. Для гостя доступні такі варіанти використання: реєстрація в системі за допомогою електронної адреси та паролю або використовуючи сервіси Google та Facebook,

вхід в систему, скидання паролю. Зареєстрованому користувачу будуть доступні такі варіанти використання: експорт інформації про проект у форматі PDF, надсилання особистого запиту на фінансування, створення, редагування, видалення та перегляд всіх створених платежів, можливість створення до 10 проектів.

На діаграмі варіантів використання більш детально та структуровано зображено прецеденти кожного з акторів. На рисунку 1.2 зображено діаграму варіантів використання системи.

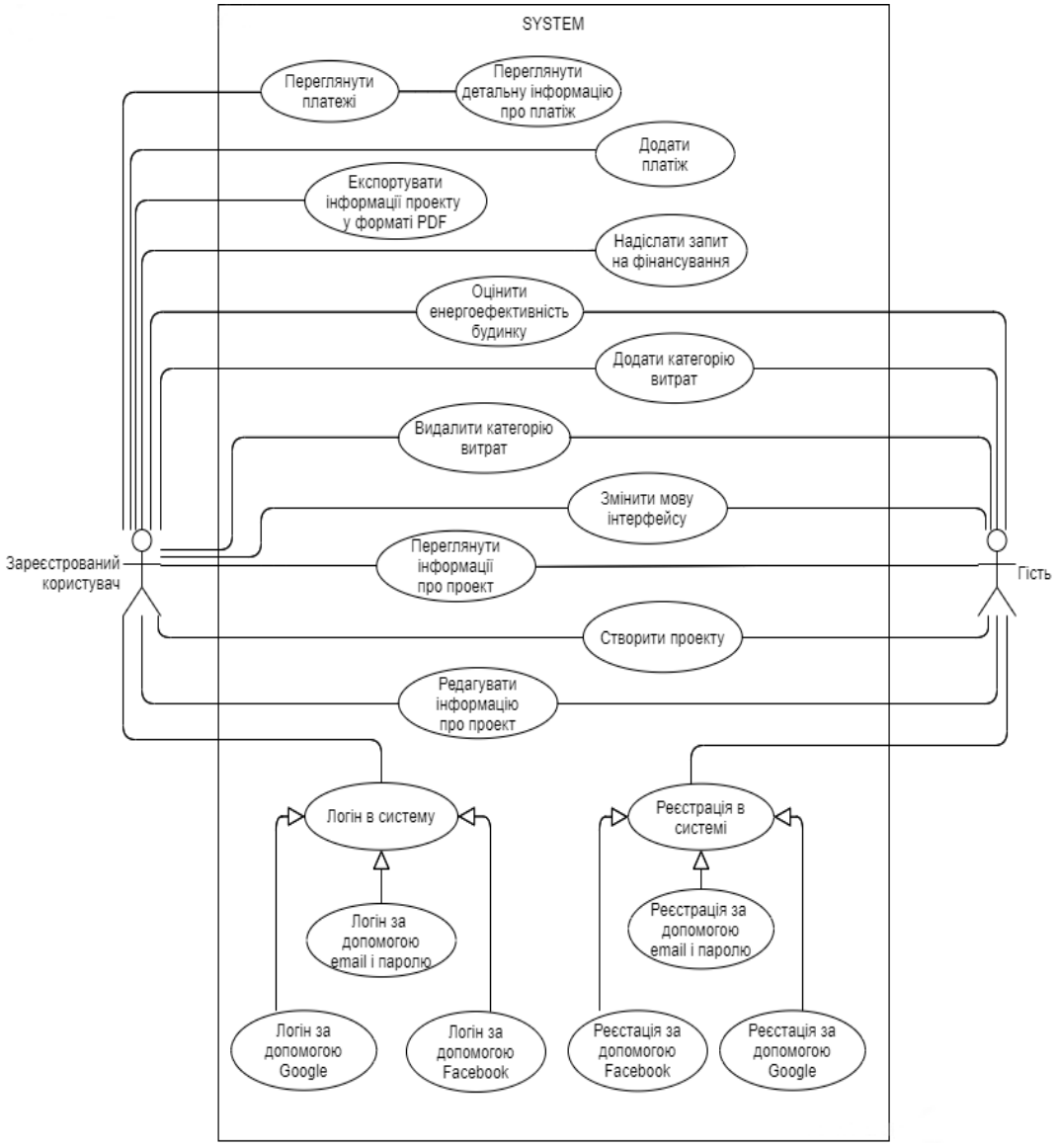



Рисунок 1.2 – Діаграма варіантів використання системи

Прецедент «Переглянути платежі» включає в себе додаткову можливість – переглянути детальну інформацію про платіж. Такі варіанти використання як «Логін в систему» та «Реєстрація» є узагальненням трьох прецедентів, які надають можливість за допомогою різних методів виконати вхід в систему.

Детальний опис основних функцій розроблювальної системи для кожного з актантів наведено в таблиці 1.1.

Таблиця 1.1 – Опис основних прецедентів системи

| Код | Найменування | Основний сценарій | Альтернативний сценарій |
|---|---|---|---|
| ЗК1 | Оцінити енерго-ефективність будинку | 1. Гість або зареєстрований користувач відкриває сторінку оцінки енергоефективності будинку.<br>2. Заповнює формі, дані якої необхідні для подальших обчислень.<br>3. Користувач натискає на кнопку подання форми.<br>4. Перенаправлення користувача на сторінку із обчисленими результатами. | 2.a. Введено не валідну інформацію<br>2.a.1. Виводиться повідомлення про те що дані не валідні<br>2.b. Не заповнено всіх обов’язкових полів<br>2.a.1. Виводиться повідомлення про те, що потрібно заповнити всі обов’язкові поля |
| ЗК2 | Створити проект | 1. Користувач переходить на сторінку створення проекту або вперше відкриває стартову сторінку.<br>2. Вводить дані у форму, яка описує детальну інформацію про проект<br>3. Натискає на кнопку подання форми<br>4. Користувач перенаправляється на сторінку інформації про план проекту | 2.a. Введено не валідну інформацію<br>2.a.1. Виводиться повідомлення про те що дані не валідні<br>2.b. Не заповнено всіх обов’язкових полів<br>2.b.1. Виводиться повідомлення про те що, потрібно заповнити всі обов’язкові поля |
| ЗК3 | Перегляд загальної інформації про проект | 1. Користувач переходить на стрінку перегляду загальної інформації проекту<br>2. На сторінці відображено загальну інформація проекту | 2.a. Сталася помилка під час отримання загальної інформації про проект.<br>2.a.1 Виводить повідомлення про помилку під час отримання інформації про проект |

Продовження таблиці 1.1

| | | | |
|---|---|---|---|
| ЗК5 | Додавання категорії витрат до проекту | 1. Користувач заходить на сторінку списку категорій витрат<br>2. Натискає на кнопку «Add category» біля бажаної категорії<br>3.Категорія додається до проекту і розраховується її приблизна вартість | 3.а. Сталася помилка під час обчислення приблизної вартості категорії витрат.<br>3.а.1 Виводить повідомлення про помилка під розрахунку приблизної вартості категорії витрат |
| ЗК4 | Редагування загальної інформації про проект проекту | 1. Користувач переходить на сторінку редагування інформації про проект<br>2. Заповнює форму редагування проекту<br>3. Натискає кнопку “Save”<br>4. Виконується перенаправ-лення на сторінку з деталь-ною інформацією про проект | 2.а. Введено не валідну інформацію<br>2.а.1. Виводиться повідомлення про те що дані не валідні |
| ЗК6 | Видалення категорій витрат з проекту | 1. Користувач заходить на сторінку списку категорій витрат<br>2. Натискає на кнопку «Delete category»<br>3. Категорія видаляється з проекту і приблизна вартість плану перераховується | 2.а. Сталася помилка під час видалення категорії витрат.<br>2.а.1. Виводиться помідомлення про помилку видалення категорії по замовчуванню |
| ЗК7 | Перегляд всіх категорій витрат проекту | 1. Користувач заходить на сторінку плану проекту, де знаходиться список всіх категорій витрат<br>2. Відображаються категорії витрат, які були додані до проекту | 2.а. Сталася помилка під час отримання категорій витрат з бекенду.<br>2.а.1. Виводиться помідомлення про помилку отримання переліку категорій проекту |
| ЗК8 | Перегляд категорії витрат проекту | 1. Користувач заходить на сторінку деталей певної категорії витрат<br>2. Відображаються деталі, які відносяться до вибраної категорії витрат | 2.а. Сталася помилка під час отримання певної категорій<br>2.а.1. Виводиться помідомлення про помилку отримання переліку категорій проекту |

Продовження таблиці 1.1

| | | | |
|---|---|---|---|
| ЗК9 | Додавання платежу до категорії витрат | 1. Користувач переходить на сторінку додавання платежу<br>2. Заповнює відповідну форму додавання платежу<br>3. Натискає кнопку “Add payment”<br>4. Відбувається перенаправлення на сторінку з детальною інформацією про платіж | 3.a. Введено не валідну або неправильну інформацію<br>3.a.1. Виводиться повідомлення про те що дані не вірні<br>3.b. Не заповнено всіх обов’язкових полів<br>2.a.1. Виводиться повідомлення про те що не всі поля заповнено |
| ЗК10 | Перегляд детальної інформації про платіж | 1. Користувач натискає на картку із платежем, для якого хоче побачити детальну інформацію.<br>2. Відкривається модальне вікно у якого відображено детальну інформацію про платіж | 2.a. Сталася помилка під час отримання детальної інформації про платіж.<br>2.a.1 Виводить повідомлення про те, що сталася помилка під час отримання детальної інформації про платіж |
| ЗК11 | Перегляд списку платежів | 1. Користувач переходить на сторінку із переліком створених платежів.<br>2. На сторінці показано платежі, які були додані до проекту | 2.a. Сталася помилка під час отримання списку платежів з бекенду.<br>2.a.1 Виводить повідомлення про те, що сталася помилка під час отримання платежів |
| ЗК12 | Експортування інформації про план створеного проекту у форматі PDF | 1. Користувач відкриває сторінку із експортом плану проекту.<br>2. Натискає на кнопку “Download”<br>3. Інформація про проект у форматі PDF експортується і завантажується на персональний комп’ютер. | 3.a. Сталась помилка під час генерації файлу PDF. 3.a.1 Виводиться повідомлення про те, що сталась помилка під час експортування файлу. |
| ЗК13 | Зміна мови інтерфейсу | 1. Користувач відкриває сторінку із налаштуваннями<br>2. Вибирає із наведеного списку бажану мову інтерфейсу<br>3. Мову системи змінено. | 3. Якщо деякого перекладу не було додано – буде виведено відповідне повідомлення про відсутність перекладу |

Дані з таблиці 1.1 надають детальну інформацію, за допомогою якої, можна проаналізувати детальний опис функціоналу кожного актора даної системи, проаналізувати перелік основний та альтернативний сценарій основних варіантів використання розроблювального застосунку.

## 1.2 Проектування програмної системи

### 1.2.1 Вибір процесу розробки

Управління розробкою програмного забезпечення (англ. Software project management) – це мистецтво планування і керування проектами з розробки програмного забезпечення, особливий вид управління проектами, в рамках якого відбувається планування, реалізація, відслідковування і контроль за проектами з розробки програмного забезпечення [6]. Кожен з підходів до розробки програмного забезпечення та існування різних методологій, кожна з яких передбачає моделювання процесу розробки програмного забезпечення у різні способи.

Для реалізації даного проекту було обрано Rapid Application Development моделлю. Життєвий цикл ПЗ методології RAD складається з чотирьох фаз:

- аналізу та планування вимог;
- проектування;
- реалізації;
- використання.

Засоби автоматизації розробки програм (Computer-Aided Software Engineering) або CASE-інструменти – це програмні продукти для проектування програм. Така система дозволяє швидко створити модель програми, а потім автоматично згенерувати програмний код. Виходить прототип – модуль, що запускається, який можна продемонструвати замовнику.

Основні принципи методології RAD:

- ітераційна розробка додатків;
- застосування CASE-засобів, що забезпечують цілісність даних;
- участь кінцевих користувачів у процесі розробки систем;
- розробка прототипів;
- тестування, яке проводиться паралельно з розробкою;
- розробка підсистем кількома, нечисленними, добре керованими командами;
- чітке планування та контроль виконання робіт.

Розробка програми з методології RAD проходить у кілька етапів. Перший – аналіз та планування. Тут визначаються цілі та завдання проекту – що й для чого робитиме додаток. Спільними зусиллями замовник та розробник виявляють ризики, встановлюють терміни та бюджет, визначають ключові моменти розробки.

Потім починається проектування користувача. На цьому етапі створюється серія працюючих прототипів програми. Кожен черговий прототип відрізняється від попереднього доповненою функціональністю, змінами дизайну та продуктивністю. Процес створення одного прототипу називається ітерацією. RAD не накладає жорстких часових рамок на тривалість однієї ітерації, але рекомендує, щоб вона була максимально швидкою.

На початку чергового циклу розробки замовник та програміст разом формулюють вимоги, яким має відповідати чергова версія. Перевага RAD у тому, що не треба заздалегідь продумувати кожну дрібницю: спочатку розробляється найзагальніша концепція, яка на наступних ітераціях доповнюватиметься й уточнюватиметься.

Прототип часто створюється нашвидкуруч, тільки для перевірки концепцій. Це нормально: якщо користувача влаштовує нова функціональність і все працює добре, в наступній ітерації розробник «відполірує» інтерфейс і код. Перфекціонізм може навіть шкодити — на черговому етапі будь-яке доопрацювання користувач може вважати невдалим. Якщо програміст прагне відразу зробити все ідеально, його зусилля виявляться марними.

Як тільки замовник дав зворотний зв'язок, цикл починається заново. Виробляється план на таку ітерацію. Якщо користувача щось не влаштувало у прототипі, на новому витку циклу зміни відкочують назад та реалізують альтернативний варіант.

Якщо замовник прийняв прототип – уточнюємо вимоги до функціональності, пророблюємо її детальніше, плануємо нову. Обговорюємо візуальні елементи та інтерфейси.

Від прототипу до прототипу програмний продукт набуває вигляду завершеної програми. Ітерації виконуються, доки не будуть реалізовані останні вимоги. Методологія RAD передбачає виконання таких ітерацій: аналіз вимог і планування, користувацьке проектування, конструювання, перемикання (див. рис. 1.3).


Рисунок 1.3 – Схема ітерацій розробки ПЗ за методологією RAD

Коли моделювання завершено, починається конструювання: автоматично згенерований код доопрацьовується і вдосконалюється.

Заключний етап розробки — впровадження. Готовий програмний продукт тестують, розвертають на користувацьких машинах, конвертують інформацію в новий формат або «заливають» у нові бази даних, підготують документацію та навчають операторів роботі в системі.

У методології RAD є і переваги, і недоліки, а також сфери застосування, в яких вона показує себе краще або гірше.

Ефективні варіанти застосування RAD. Якщо проект легко ділиться на незалежні чи слабозв'язані модулі. Розробку в такому випадку можна вести паралельно кількома командами, кожна з яких збиратиме прототип тільки одного модуля. Наприкінці ітерації чи всієї роботи над програмою модулі об'єднують у цільну програму.

Якщо вимоги до програмного забезпечення швидко змінюються. RAD - відмінний вибір, коли замовник розуміє, що програма потрібна якнайшвидше, але до кінця роботи над нею частина специфікацій напевно зміниться.

У разі обмеженого бюджету. RAD гарантує, що замовник отримає продукт, який виконує поставлені завдання, навіть якщо раптово закінчаться гроші.

Коли у користувача немає чіткого уявлення, як має виглядати та працювати продукт. Оскільки програму створюють невеликими ітераціями, під час яких специфікації та вимоги постійно уточнюються, у результаті замовник отримує продукт, який відповідає його побажанням. Але краще загалом спереду сформулювати бізнес-мети та завдання для застосування.

Функціональність, яка потрібна замовнику «ще вчора», можна розробити в першу чергу, і використовувати навіть якщо інші частини програми ще не готові.

Недоліки методології RAD:

- RAD застосовується для великих команд;
- RAD залежить від залучення замовника до роботи. Якщо він не може взяти участь у черговому обговоренні проекту, робота може призупинитись.

Основні переваги методології:

- розробка виконується швидко та дешево;
- RAD забезпечує прийнятний для користувача рівень якості;
- користувач отримує саме ту функціональність, яку хоче;
- користувач може оперативно внести зміни до проекту.

Методологія як і раніше, використовується в розробці програмного забезпечення і здавати свої позиції не збирається. Адже для методології головне не вік, а ефективність.

### 1.2.2 Побудова UML діаграми діяльності програмної системи

Діаграма діяльності (англ. activity diagram) — в UML, візуальне представлення графу діяльностей [7]. Моделювання діаграми діяльності підкреслює послідовність і умови координації поведінки нижчого рівня, проте не показує, які класифікатори володіють цією поведінкою. Їх зазвичай називають моделями потоку управління і потоку об'єктів. Дії, координовані моделями дій, можуть бути ініційовані після завершення виконання інших дій, оскільки об'єкти і дані стають доступними або відбуваються зовнішні події.

Діаграми діяльності зазвичай використовуються для моделювання бізнес-процесів, для моделювання логіки, що охоплюється одним варіантом використання або сценарієм використання, або для моделювання докладної логіки бізнес-правила. Хоча діаграми діяльності UML потенційно можуть моделювати внутрішню логіку складної операції, було б набагато краще просто переписати операцію таким чином, щоб вона була досить простою, щоб вам не потрібна діаграма дій. У багатьох відношеннях діаграми активності UML є об'єктно-орієнтованим еквівалентом блок-схем і діаграм потоків даних з структурованої розробки.

Мета діаграми діяльності полягає в тому, щоб змоделювати процедурний потік дій, які є частиною більшої діяльності. У проектах, в яких присутні варіанти використання, діаграми діяльності можуть моделювати конкретний варіант використання на більш детальному рівні. Однак діаграми даного типу можна використовувати незалежно від варіантів використання для моделювання функції бізнес-рівня. Діаграму діяльності також можна використовувати для моделювання функцій системного рівня. Оскільки він моделює процедурний потік, діаграма фокусується на послідовності виконання дій і умовах, які запускають або захищають ці дії. Діаграма активності також орієнтована тільки на внутрішні дії активності, а не на дії, які викликають активність в їх потоці процесів або запускають активність відповідно до будь-якої події.

Хоча діаграми послідовностей UML можуть відображати ту ж інформацію, що і діаграми діяльності, проте вважається, що остання найкраще підходить для моделювання функцій бізнес-рівня. Це пов'язано з тим, що в діаграмі діяльності показуються всі потенційні потоки послідовності в дії, в той час як діаграма послідовності зазвичай показує тільки один потік дії. Крім того, бізнес-менеджери та персонал бізнес-процесів, віддають перевагу діаграмам діяльності діаграмам послідовності дій – діаграма діяльності виглядає менш "технічною" і, отже, менш складною для ділових людей. Крім того, бізнес-менеджери звикли бачити блок-схеми, тому "зовнішній вигляд" діаграми діяльності знаком.

Діаграми діяльності є однією з найдоступніших діаграм UML, оскільки в них використовуються символи, подібні до широко відомої блок-схеми; тому вони корисні для опису процесів для широкої аудиторії. Насправді діаграми діяльності мають свої корені в блок-схемах, а також у діаграмах станів UML, діаграмах потоків даних.

Спроектовану діаграму діяльності системи зображено на рисунку 1.4.

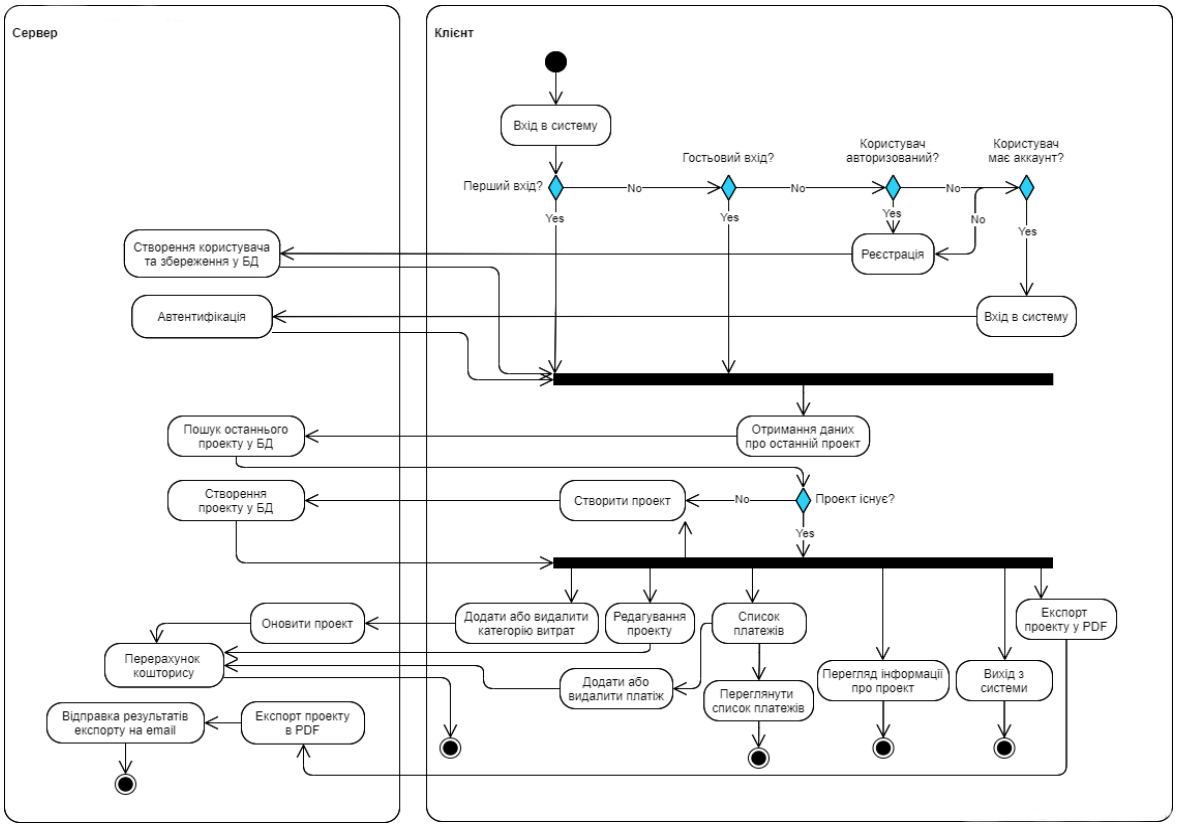



Рисунок 1.4 – Діаграма діяльності

Дана діаграма відображає доступні дії користувача в розроблювальній системі. Початок роботи починається із входу в додаток, де відбувається розгалуження при визначенні стану системи. Перший можливий варіант є коли система створює користувачу гостьовий профіль, який користувач зможе користуватись доки не вирішить створити обліковий запис із збереженням всього прогресу, або закрити вкладку, при цьому втративши всі незбережені дані. Другий варіант є актуальним тоді, коли користувач вже має створений обліковий запис в системі та виконує авторизацію. Після чого виконується активність «отримати останній проект», під час виконання якої виконується запит до сервера, який повертає необхідну інформацію про останній створений проект. У випадку, якщо жодного проекту не було знайдено, відбудеться перенаправлення користувача на активність додавання нового проекту, який в подальшому буде відправлений на сервер та записаний в базу даних. Користувач має можливість доступу тільки до тих проектів, які створив сам.

Після створення проекту користувач має доступ до таких активностей: редагування детальної інформації про проект; додавання категорії витрат; перегляд плану проекту; експортування плану у форматі PDF; додавання та редагування платежів; перегляд створених платежів.

### 1.2.3 Моделювання архітектури системи

Розроблювальна веб-орієнтована система є клієнт-серверним застосунком. Клієнтом являється комп’ютер на стороні користувача, який виконує запити до сервера. Розробка додатку відбуватиметься за принципом трирівневої клієнт-серверної архітектури.

Трирівнева архітектура клієнт-сервер також відома як багаторівнева архітектура і сигналізує про введення середнього рівня для посередництва між клієнтами і серверами. Середній рівень існує між інтерфейсом користувача на

стороні клієнта та системою керування базами даних (СУБД) на стороні сервера. Цей третій рівень виконує управління процесами, що включає реалізацію бізнес-логіки та правил. Трирівневі моделі можуть вмістити сотні користувачів. Він приховує від користувача складність розподілу процесів, водночас може виконувати складні завдання за допомогою черги повідомлень, реалізації додатків і постановки даних або зберігання даних перед завантаженням у сховище даних.

Як і в дворівневих архітектурах, верхній рівень – це інтерфейс системи користувача (клієнт), а нижній – управління базою даних. Рівень керування базою даних забезпечує узгодженість даних за допомогою таких функцій, як блокування та реплікація даних. Блокування даних також називають блокуванням файлів або записів. Це функція СУБД «перший прийшов, перший обслуговується», що використовується для керування даними та оновленнями в багатокористувацькому середовищі. Перший користувач, який отримав доступ до файлу або запису, відмовляє іншим користувачам у доступі або «блокує їх». Він знову відкривається і стає доступним для інших користувачів після завершення оновлення.

Середній рівень також називають сервером додатків. Він містить централізовану логіку обробки, яка полегшує управління та адміністрування. Локалізація функціональності системи на середньому рівні дає змогу обробляти зміни та оновлення одноразово та поширюватися по всій мережі, доступній як клієнтам, так і серверам. Іноді середній ярус поділяється на два або більше підрозділів з різними функціями. Це робить її багатошаровою моделлю. Використовуючи мову сценаріїв, вбудовану в HTML, веб-сервери діють як рівні перекладу, які забезпечують зв’язок між клієнтськими і серверними рівнями.

Цей рівень отримує запити від клієнтів і генерує відповіді HTML після запиту від серверів баз даних Однією з головних переваг трирівневої архітектури є можливість розділяти програмне забезпечення та «перетягувати» модулі на різні комп’ютери в мережі.

Representational State Transfer (REST) – це архітектура зв’язку клієнт-сервер, яка створює розділення проблем між ресурсами даних та інтерфейсами користувача. Послуги, які реалізують REST, називаються RESTful. Сервер керує

ресурсами даних. Клієнти можуть вільно впроваджувати інтерфейс користувача будь-яким способом або мовою. Архітектура REST не стосується того, як реалізуються інтерфейси користувача. Вона стосується лише підтримки стану програми між клієнтом і сервером.

Кожна програма є унікальною, проте більшість поділяє деякі спільні проблеми, такі як інфраструктура хостингу, управління ресурсами, презентація та поведінка інтерфейсу. Інфраструктура може мати багато різновидів і мати багато різних механізмів кешування. Як правило, це складається з таких компонентів:

- сховище даних;
- віртуальна приватна мережа (VPN) або брандмауер (щоб захистити сховище даних від несанкціонованого доступу);
- JSON RESTful Web Service Layer;
- різні сторонні API;
- сервер додатків/система керування вмістом (CMS) для маршрутизації запитів і доставки сторінок клієнту;
- мережа доставки статичного вмісту (CDN).

Поєднання всіх перелічених компонентів зображено на рисунку 1.5.

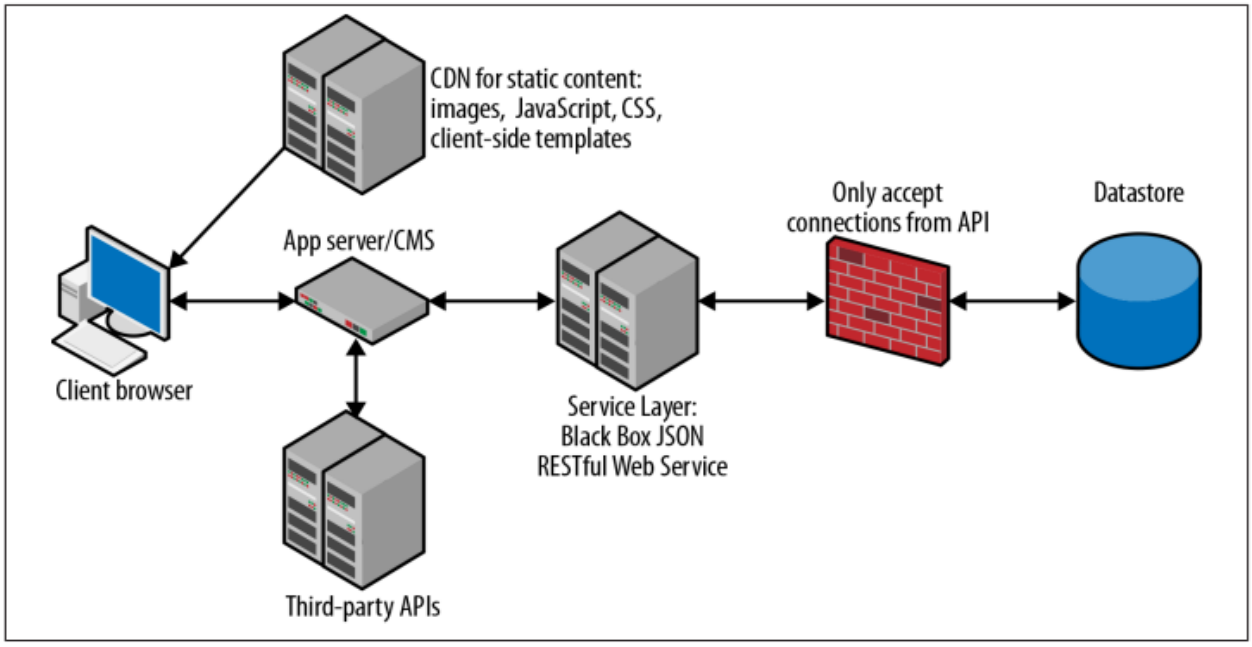



Рисунок 1.5 – Інфраструктура трирівненого клієнт-серверного додатку

Більшість із цих компонентів зрозумілі самі собою, але є деякі важливі моменти, про які потрібно зауважити, що стосуються зберігання та передачі даних програми. Сховище даних є місцем для зберігання даних програми. Зазвичай це система управління реляційною базою даних (RDBMS) з API мови структурованих запитів (SQL), але популярність рішень NoSQL зростає. У майбутньому, ймовірно, що багато програм використовуватимуть комбінацію обох.

## 1.3 Конструювання системи

### 1.3.1 Вибір мови програмування та технологій розробки

Для розробки системи було вибрано мову програмування JavaScript. Це полегшить процес розробки системи та її підтримки в подальшому, оскільки система складається з кількох компонентів це також спростить процес взаємодії інженерів, які працюють над різними її модулями. Мова програмування JavaScript використовується як для забезпечення імплементації сервера, за допомогою оточення Node.js, який побудований на JavaScript рушієві Chrome V8, так і для виконання коду в браузері для надання інтерактивності сторінкам.

JavaScript – це мова сценаріїв на стороні клієнта. Вона використовується для поліпшення функціональності веб-сайту або веб-сторінки. Крім того, вона використовується в поєднанні з HTML і CSS для створення адаптивних веб-сайтів [8].

JavaScript є однією з багатьох мов сценаріїв на стороні клієнта (наприклад, VBScript, PerlScript, Jscript, ActionScript), які існують в Інтернеті. Однак дана мова є найбільш популярною і широко використовуваною. Через її широке використання вона створила безліч фреймворків. Фреймворки існують для полегшення кодування. Вони також покращують і розширюють функціональність мов.

З іншого боку, дана мова сценаріїв лежить в основі багатьох функцій веб-сайту та додаткових функцій в Інтернеті. Можна з упевненістю сказати, що в порівнянні з іншими мовами JavaScript є найбільш використовуваним в світі. Оскільки майже кожен веб-сайт використовує цю мову програмування на кожній своїй сторінці, без сумніву.

Крім того, JavaScript є кросплатформенним і є корисним інструментом для створення імпровізованих програм і макросів. Завдяки масовій доступності браузерів на кожному комп'ютері і інтелектуальних пристроях є можливість легко створити хорошу програму з використанням редактора для редагування тексту і веб-браузера.

Незважаючи на стандартизованість, потрібно бути обережним, що браузери можуть вести себе або працювати по-різному в деяких елементах коду. Крім того, JavaScript також має версію сценаріїв на стороні сервера. Мова надає більше можливостей для сценаріїв на стороні клієнта.

Під час входу на на веб-сторінку в Інтернеті, дві сутності відіграють важливу роль у наданні вам необхідного контенту. По-перше, це клієнт, а по-друге, це сервер. Клієнт - це машина, комп'ютер або пристрій, який використовується для доступу в інтернет. Користувацький браузер також є частиною "клієнта". Сервер - це комп'ютер, який зберігає і надає клієнтам необхідний їм контент. З'єднання між клієнтом і сервером зазвичай забезпечується підключенням до Інтернету.

Місце розташування і сам сервер змінюються в залежності від необхідного запитуваного клієнтом контенту. Сервер може перебувати в мережі, в межах локального підключення користувача або навіть може бути користувацьким комп'ютером.

Сценарій подій, які відбуваються одна за одною під час входу на звичайну веб-сторінку:

- браузер відправить запит на сервер для отримання вмісту веб-сторінки;
- сервер обробить запит і знайде необхідний контент;
- сервер відправить вміст назад клієнту;

– клієнт обробить вміст і відобразить його на екрані.

За досить короткий проміжок часу у кілька років, розробка веб-додатків кардинально змінила свій вигляд. На сьогоднішній день існує багато варіантів, і новачки часто задаються питанням, який стек технологій вибрати. Це стосується не тільки широкого стеку (різних рівнів або використовуваних технологій), але й інструментів, які допомагають у розробці.

Будь-який веб-додаток створюється з використанням декількох технологій. Комбінація цих технологій називається "стеком". У міру розвитку веб-розробки і появи інтерактивності на перший план все більш популярними ставали односторінкові додатки (SPA). Це призвело до зростання числа інтерфейсних фреймворків, оскільки більша частина роботи була виконана на стороні клієнта.

Для розроблюваного додатку важливими елементами є продуктивність і простота у користуванні. Під час аналізу широко використовуваних стеків технологій вибір було зупинено на MERN стеку, який буде в змозі забезпечити всі потреби розробника та користувача.

React – це бібліотека JavaScript, яка дозволяє створювати користувальницький інтерфейс як для мобільних, так і для веб-додатків. Дана бібліотека легко інтегрується з іншими фреймворками і бібліотеками JavaScript і включає невеликі, багаторазово використовувані фрагменти коду, так звані компоненти. Завдяки своїй високій модульності бібліотеки компонентів React не тільки оптимізують розробку користувальницького інтерфейсу, але і забезпечують виняткову гнучкість.

Бібліотека React допомагає не тільки створювати функціональні та зручні у користуванні додатки, але й дозволяє робити це швидше, простіше і з меншою кількістю коду, ніж інші бібліотеки. React - це визначальний компонент MERN стеку.

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript [9].

Платформа являє собою кросплатформенне середовище виконання з відкритим вихідним кодом, що використовується для розробки серверних веб-

додатків. Node.js додатки написані на JavaScript і можуть запускатися в різних операційних системах.

Node.js заснований на архітектурі, керованій подіями, і не блокуючому API введення та виводу, який призначений для оптимізації масштабованості програми для веб-додатків реального часу.

Протягом тривалого періоду часу всі фреймворки, доступні для веб-розробки, були засновані на моделі без збереження стану. Модель без стану - це коли дані, згенеровані за один сеанс. Наприклад, інформація про налаштування користувача та події, що відбулися не зберігається для використання в наступному сеансі з цим користувачем. Було виконано великий об’єм роботу, щоб зберегти інформацію про сеанс між запитами для користувача. Але з Node.js нарешті з'явився спосіб для веб-додатків мати двосторонні з'єднання в режимі реального часу, при яких і клієнт, і сервер можуть ініціювати зв'язок, дозволяючи їм вільно обмінюватися даними.

Express – це середовище виконання, яке може запускати JavaScript, платформа, яка спрощує завдання написання серверного коду. Специфікація відповідності заснована на регулярному вираженні і є дуже гнучкою, як і більшість інших платформ веб-додатків. Експрес аналізує URL-адресу запиту, заголовки і параметри для вас. Що стосується відповіді, як і очікувалося, фреймворк має всі функції, необхідні для веб-додатків. Це включає в себе налаштування кодів відповіді, налаштування файлів cookie, відправку користувальницьких заголовків. Крім того, є можливість написати проміжне програмне забезпечення Express, яке являє собою користувальницькі фрагменти коду, які можуть бути вставлені в будь-який шлях обробки запитів або відповідей для досягнення загальних функцій, таких як ведення журналу, аутентифікація тощо. Для SPA нам не потрібно буде використовувати серверний механізм шаблонів. Це пов'язано з тим, що вся генерація динамічного контенту виконується на клієнті, а веб-сервер обслуговує тільки статичні файли і дані за допомогою викликів API. Особливо в стеку MERN генерація сторінок обробляється самим React на стороні сервера.

MongoDB – це документно-орієнтована база даних NoSQL, з гнучкою схемою і мовою запитів на основі JSON. Одиницею зберігання є документ або об'єкт. Кілька документів зберігаються в колекціях. Кожен документ в колекції має унікальний ідентифікатор, який індексується автоматично, та за яким до нього можна отримати доступ до документу. MongoDB має можливість індексувати глибоко вкладені поля, чого не можуть робити реляційні бази даних. Недоліком є те, що дані зберігаються денормалізованими. Це означає, що дані іноді дублюються, вимагаючи більше місця для зберігання. Крім того, такі речі, як перейменування основного імені запису, означали б проходження через всю базу даних.

NPM – це менеджер пакетів за замовчуванням для Node.js. Його можна використовувати для встановлення сторонніх бібліотек, а також для управління залежностями між ними. Реєстр NPM є загальнодоступним сховищем всіх модулів, опублікованих користувачами з метою спільного використання. Використання npm з командного рядка для встановлення, видалення або оновлення пакетів.

Npm відомий як найбільший у світі реєстр програмного забезпечення. Розробники з відкритим кодом у всьому світі використовують npm для публікації та обміну вихідним кодом.

Хоча NPM починався як сховище для Node.js модулів, він швидко перетворився в менеджер пакетів для доставки інших модулів на основі JavaScript, зокрема тих, які можна використовувати в браузері. Насправді, незважаючи на те, що React в основному є кодом на стороні клієнта і може бути включений безпосередньо в HTML у вигляді файлу скрипту, рекомендується замість цього встановлювати React через NPM. Але після встановлення у вигляді пакету потрібен інструмент для того, щоб об'єднати весь код, який може бути включений в HTML, щоб браузер міг отримати доступ до коду. Для цього існують інструменти збірки, такі як browserify або webpack, які можуть об'єднувати модулі, а також сторонні бібліотеки в пакет, який може бути включений в HTML.

### 1.3.2 Вибір СКБД та опис її фізичної моделі

База даних (англ. database) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами. Ця сукупність підтримує щонайменше одну з областей застосування [10].

Розроблювальна веб-орієнтована система використовує документно-орінтовану СКБД MongoDB. MongoDB – це база даних документів, що означає, що еквівалентом запису є документ або об’єкт. У реляційній базі даних потрібно використовувати рядки та стовпці, тоді як у базі даних документів цілий об’єкт можна записати як документ.

Для простих об’єктів це може здатися нічим не відрізнятися від реляційної бази даних. Але припустимо, у нас є об’єкти з вкладеними об’єктами і масивами. Тепер, коли використовується реляційна база даних, зазвичай потрібно кілька таблиць.

Значення полів можуть включати інші документи, масиви документів. Документи MongoDB подібні до об’єктів JSON, тому їх легко вважати об’єктами JavaScript. У порівнянні з об’єктом JSON, документ MongoDB підтримує не тільки примітивні типи даних, логічні значення, числа та рядки, а й інші поширені типи даних, такі як дати, позначки часу, регулярні вирази та двійкові дані. Ми навіть можемо зберігати функції JavaScript як поля документу.

MongoDB колекція подібна до таблиці в реляційній базі даних. Це набір документів, і ми отримуємо доступ до кожного документа через колекцію. Як і в реляційній базі даних, ми можемо мати первинний ключ і індекси в колекції. Але є кілька відмінностей.

Первинний ключ є обов'язковим у MongoDB, і він має зарезервоване ім'я поля _id. Навіть якщо поле _id не було вказано при створенні документу, MongoDB автоматично генерує унікальний ключ для кожного документа. Найчастіше автогенерація використовується як є, оскільки вона зручна і гарантовано створює унікальні ключі, навіть коли кілька клієнтів одночасно записують дані в базу даних.

Поле _id автоматично індексується. Крім цього, індекси можуть бути створені для інших полів, включаючи поля у вбудованих документах і поля масиву. Індекси використовуються для ефективного доступу до підмножини документів у колекції.

На відміну від реляційної бази даних, MongoDB не вимагає, щоб визначення схеми для колекції. Єдина вимога полягає в тому, що всі документи в колекції повинні мати унікальний _id, але фактичні документи можуть мати абсолютно різні поля. На практиці, однак, всі документи в колекції мають однакові поля. Хоча гнучка схема може здатися дуже зручною для зміни схеми на початкових етапах програми, це може викликати проблеми, якщо в коді програми немає будь-якої перевірки схеми.

На відміну від універсальної SQL в реляційній базі даних, мова запитів MongoDB складається з методів для виконання різних операцій. Основними методами операцій читання і запису є методи CRUD. Інші методи включають агрегування, текстовий пошук і геопросторові запити.

Всі методи працюють з колекцією і приймають параметри в якості об'єктів JavaScript, які визначають деталі операції. Кожен метод має свою власну специфікацію. Наприклад, щоб вставити документ, єдиний параметр, який нам потрібен, – це сам документ. Для запиту параметри являють собою специфікацію відповідності і список повернутих полів.

На відміну від реляційних баз даних, не існує методу, який міг би працювати з декількома колекціями одночасно. Всі методи працюють тільки з однією колекцією одночасно. Якщо є необхідність об'єднати результат декількох колекцій, кожна колекція повинна запитуватися окремо і оброблятися клієнтом. У реляційній базі даних можна використовувати об'єднання для об'єднання таблиць з використанням полів, загальних для таблиць, щоб результат включав вміст обох таблиць. Це дозволяє масштабувати бази даних NoSQL за допомогою сегментів або декількох серверів для розповсюдження документів, що входять в одну колекцію.

Крім того, на відміну від реляційних баз даних, MongoDB заохочує денормалізацію, тобто зберігання пов'язаних частин документа у вигляді

вбудованих вкладених документів, а не у вигляді окремих колекцій в реляційній базі даних.

Традиційні платформи СУБД забезпечують масштабованість з використанням підходу масштабування, який вимагає більш швидкого сервера для підвищення продуктивності. Наступні проблеми в системах СУБД призвели до того, що MongoDB та інші бази даних NoSQL були спроектовані так, як вони спроектовані:

– для масштабування бази даних СУБД необхідно зв'язати дані, доступні в двох або більше системах, щоб повідомити про результати. Цього важко досягти в системах СУБД, оскільки вони призначені для роботи, коли всі дані доступні для спільного використання обчислення. Таким чином, дані повинні бути доступні для обробки в одному місці;

– у випадку декількох активних серверів, коли обидва оновлюються з декількох джерел, виникає проблема з визначенням правильного оновлення;

– коли додаток намагається прочитати дані з другого сервера, і інформація була оновлена на першому сервері, але ще не синхронізована з другим сервером, тоді інформація, що повертається може бути застарілою.

Розробники MongoDB прийняли рішення використовувати нереляційний підхід до вирішення цих проблем. Запити в MongoDB засновані на ключах в документі, тому документи можуть бути розподілені по декількох серверах. Запит кожного сервера означає, що він перевірить свій власний набір документів і поверне результат. Це забезпечує лінійну масштабованість і підвищену продуктивність.

MongoDB має первинну-вторинну реплікацію, де первинна приймає запити на запис. Якщо необхідно підвищити продуктивність запису, то можна використовувати сегментування; це розподіляє дані по декількох машинах і дозволяє цим декільком машинам оновлювати різні частини наборів даних. Сегментування в MongoDB виконується автоматично; у міру додавання нових машин дані розподіляються автоматично.

MongoDB використовує сховище документів на основі JSON. JSON / BSON пропонує модель без схеми, яка забезпечує гнучкість з точки зору проектування бази даних. На відміну від СУБД, в схему можна вносити зміни без проблем. Цей дизайн також забезпечує високу продуктивність, забезпечуючи внутрішню групування відповідних даних і полегшуючи їх пошук.

MongoDB орієнтована на документи СУБД, в якій дані зберігаються у вигляді документів. Він не підтримує об'єднання і не має повністю узагальнених транзакцій. Тим не менш, він забезпечує підтримку вторинних індексів, дозволяє користувачам виконувати запити з використанням документів запитів і забезпечує підтримку атомарних оновлень на рівні кожного документа. Він надає набір реплік, який являє собою форму реплікації з автоматичним відпрацюванням відмови, і має вбудоване горизонтальне масштабування.

Визначимо особливості, якими MongoDB відрізняється від SQL:

1. MongoDB використовує документи для зберігання своїх даних, які пропонують гнучку схему. Це дозволяє зберігати вкладені або багатозначні поля, такі як масиви, хеші і т.д. Реляційні СУБД пропонують фіксовану схему, в якій значення стовпця повинно мати аналогічний тип даних.

2. MongoDB не підтримує операції з’єднання, як в SQL. Однак вона має обхідний шлях для вирішення цієї проблеми, це дозволяє зберігати всі відповідні дані разом в одному документі, уникаючи на периферії використання з'єднань.

3. MongoDB не забезпечує підтримку транзакцій таким же чином, як SQL. Однак, це гарантує атомарність на рівні документа.

Більшість баз даних змушують вас використовувати важкі обгортки, такі як ORM, щоб отримати дані в об’єктній формі для використання в програмах. Рішення MongoDB зберігати та представляти дані у форматі документа означає, що ви можете отримати до них доступ з будь-якої мови, у структурах даних, які є рідними для цієї мови.

Здійснивши аналіз предметної області, було визначено основні сутності, які необхідно відобразити відповідним чином у базі даних. База даних міститиме такі колекції: $CO_2$ calculator, Project, Plan, Category, User, Payment, Financing request.

Розглянемо більш детально описані вище основні сутності розроблювальної системи. Однією з головних сутностей є сутність калькулятора енергоефективності будинку. Дана сутність містить дані користувача з такою інформацією як: опалювальна житлова площа, тип будинку, енергетичний стандарт будинку, проведені ремонтні роботи, тип опалювальної системи, чи є сонячна теплова система та інші параметри. Схему калькулятора енергоефективності будинку наведено на рисунку 1.6.

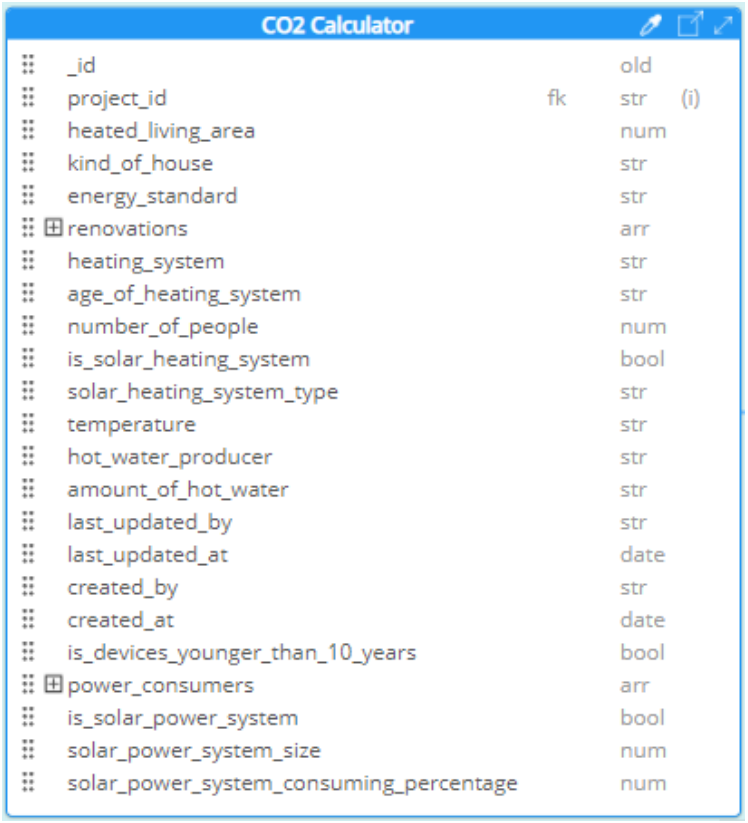



Рисунок 1.6 – Схема сутності «$CO_2$ калькулятор будинку»

Сутність $CO_2$ калькулятор будинку містить такі поля:

- _id – унікальний ідентифікатор;
- project_id – зовнішній ключ, унікальний ідентифікатор проекту;
- heated_living_area – опалювальна житлова площа;
- kind_of_house – тип будинку;
- age_of_heating_system – вік опалювальної системи;

– number_of_people – кількість людей, яка проживає в будинк;
– temperature – яка переважна температура в будинку;
– hot_water_producer – який виробник гарячої води;
– amount_of_hot_water – як багато гарячої води необхідно;
– power_consumers – незвичні споживачі енергії;
– is_solar_power_system – наявність сонячної енергетичної системи;
– solar_power_system_type – тип сонячної енергетичної системи;
– solar_power_system_consuming_percentage – відсоток споживання від сонячної енергетичної системи;
– created_at – дата створення;
– last_updated_at – дата останнього оновлення;
– created_by – зовнішній унікальний ідентифікатор, який вказує на користувача, який здійснив обчислення;

Приклад екземпляру сутності $CO_2$ калькулятор наведено на рисунку 1.7.


```
{
    "heated_living_area": 128,
    "kind_of_house": "bungalow_or_complex_floor_plan",
    "energy_standard": "between_1976_and_1990",
    "renovations": ["new_windows", "insulation_facade"],
    "solar_heating_system_type": null,
    "number_of_people": 4,
    "heating_system": "natural_gas",
    "is_solar_heating_system": false,
    "temperature": "between_21_and_23",
    "hot_water_producer": "electric_boiler",
    "amount_of_hot_water": "medium",
    "is_devices_younger_than_10_years": true,
    "power_consumers": ["swimming_pool", "sauna"],
    "is_solar_power_system": false,
    "solar_power_system_size": null,
    "solar_power_system_consuming_percentage": null,
    "age_of_heating_system": "between_10_and_20_years",
        "created_at": "2021-08-06T14:08:11.761+00:00",
    "last_updated_at": "2021-08-08T16:03:19.353+00:00",
    "created_by": "f450a321-88d4-4609-8617-17a4a570284a"
}
```


Рисунок 1.7 – Лістинг коду екземпляру сутності $CO_2$ калькулятор

Наступною однією з головних сутностей системи є сутність проекту, оскільки головною метою розроблювального проекту є створення проекту планування витрат у сфері будівництва. Дана сутність містить в собі детальну інформацію про планований проект користувача. Схему проекту зображено на рисунку 1.8.

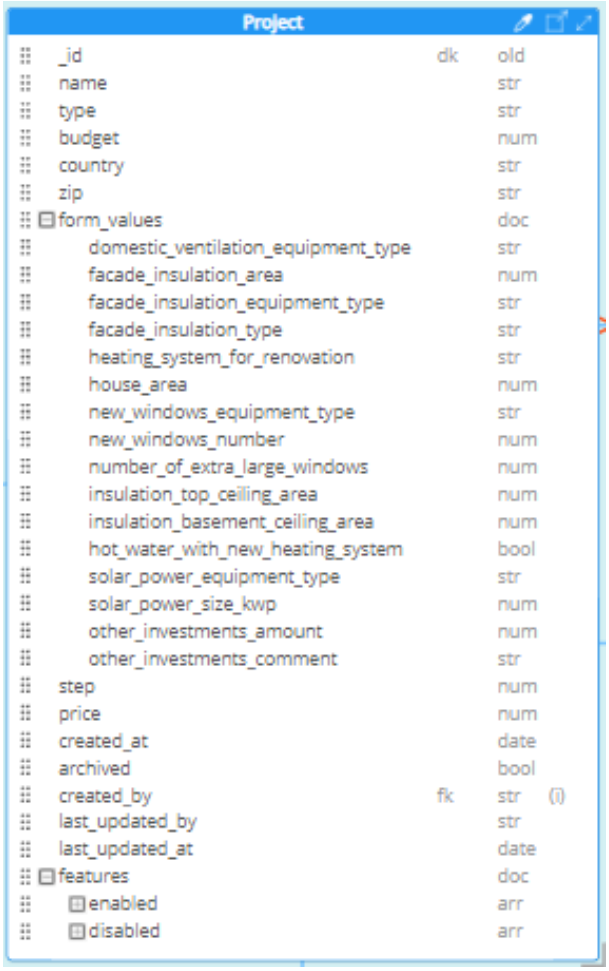



Рисунок 1.8 – Схема сутності проекту

Дана сутність проекту містить такі дані як:

– name – назва проекту;
– type – тип проекту;
– budget – бюджет користувача на планований проект;
– country – назва країни;
– form_values.domestic_ventilation_equipment_type – тип побутової вентиляція;

- form_values.facade_insulation_area – площа утеплення фасаду;
- form_values.facade_insulation_equipment_type – тип обладнання для утеплення фасаду;
- form_values.facade_insulation_type – тип утеплення фасаду;
- form_values.heating_system_for_renovation – планований тип опалювальної системи;
- form_values.house_area – загальна площа будинку;
- form_values.new_windows_equipment_type – тип обладення нових вікон;
- form_values.new_windows_number – кількість звичайних вікон;
- form_values.number_of_extra_large_windows – кількість великих вікон;
- form_values.insulation_top_ceiling_area – утеплення стелі верхнього поверху;
- form_values.insulaiton_basement_ceiling_area – утеплення стелі нижнього поверху;
- form_values.hot_water_with_new_heating_system – зміна виробника гарячої вода з новою системою опалення;
- form_values.solar_power_equipment_type – тип обладнання сонячних електростанцій;
- form_values.solar_power_size_kwp – розмір сонячних електростанцій в kwp;
- form_values.other_investments_amount – сума інвестицій на інші витрати;
- form_values.other_investments_comment – коментар на інші інвестиції;
- step – крок;
- price – обчислена вартість проекту;
- created_at – дата створення проекту;
- archived – значення, яке вказує чи проект архівовано;
- created_by – зовнішній ключ, який вказує на ідентифікатор користувача, який створив проект;
- last_updated_at – дата останнього оновлення;
- features.enabled – перелік вибраних категорій витрат, для планування;
- features.disabled – перелік виключених категорій витрат;

Приклад екземпляру сутності проекту наведено на рисунку 1.9.


```
    "_id": "610d47d22ead580004294ed3",
    "name": "Mein Wohntraum",
    "type": "renovation_house",
    "budget": 50000,
    "country": "AT",
    "step": 5,
    "form_values": {
        "domestic_ventilation_equipment_type": "premium",
        "facade_insulation_area": 236,
        "facade_insulation_equipment_type": "premium",
        "facade_insulation_type": "single",
        "heating_system_for_renovation": "natural_gas",
        "house_area": 100,
        "new_windows_equipment_type": "premium",
        "new_windows_number": 2,
        "number_of_extra_large_windows": 1,
        "number_of_floors": 2,
        "other_investments_amount": 10000,
        "other_investments_comment": "comment",
        "renewal_of_heating_system_equipment_type": "premium",
        "solar_power_equipment_type": "standard",
        "solar_power_size_kwp": 4,
        "hot_water_with_new_heating_system": true,
        "insulation_basement_ceiling_area": 50,
        "insulation_top_ceiling_area": 50
    },
    "meta": { "clientIP": "11.148.114.15","domain": "localhost:3000" },
    "country_index": 108.5,
    "created_by": "e2d6bc2e-27c6-4010-b66d-d520beabeba4",
"features": {"enabled": ["other_investments", "solar_power_system", "new_windows",
        ], "disabled": [] },
```


Рисунок 1.9 – Лістинг коду екземпляру сутності проекту

Наступною сутністю розроблювальної системи є категорія планованих користувачем витрат. Одна категорія описує один елемент плану проекту. Обчислення вартості кожної з категорій здійснюються за допомогою формул, які зберігаються в кожній із сутностей. Однією з найважливіших функцій розроблювальної системи є гнучкість. Тому перед нами постає завдання реалізувати метод задання формули так, щоб алгоритм обчислення ціни кожної з категорій можна було б регулювати шляхом зміни значень коефіцієнтів у базі даних. Тому, було вирішено реалізувати інтерпретатор виразів, щоб можна було виразити залежності між полями як рядок. Отже, це надасть нам можливість

динамічно описувати елементи плану проекту без повторення оновлення публікації.

Сутність категорії містить такі поля:

- price – стрічка формули, яка використовується для розрахунку ціни однієї категорії на основі даних з проекту;
- name – ключ із назвою категорії, який додаватиметься в масив features сутності проекту.
- label – ключ до назви категорії, який використовуватиметься в інтерфейсі із інтернаціоналізацією;
- user_price – значення, яке вказує чи може користувач перезаписувати розраховану ціну категорії;
- equipment_types – тип планованого обладнання, програмно представляє собою об’єкт із ключами, які позначають якість обладнання (standard, premium, superior);
- price_per_unit – значення вартості одиниці відповідного обладнання;
- description – ключ до опису категорії;
- unit – одиниця вимірювання.

Схему сутності категорії витрат наведено на рисунку 1.10.

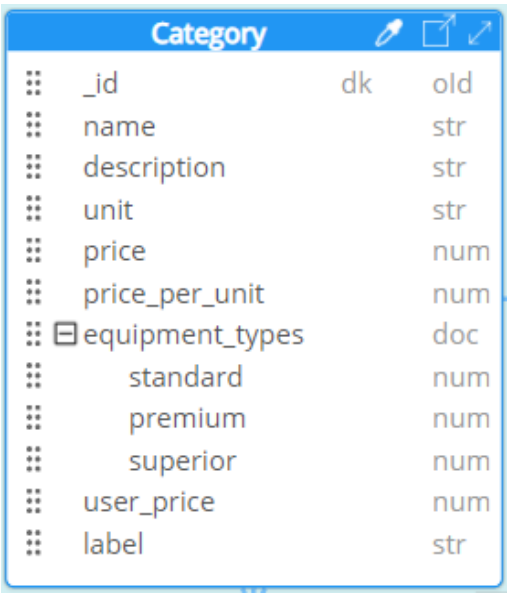



Рисунок 1.10 – Схема сутності категорії витрат

Приклад екземпляру сутності категорії витрат наведено на рисунку 1.11.

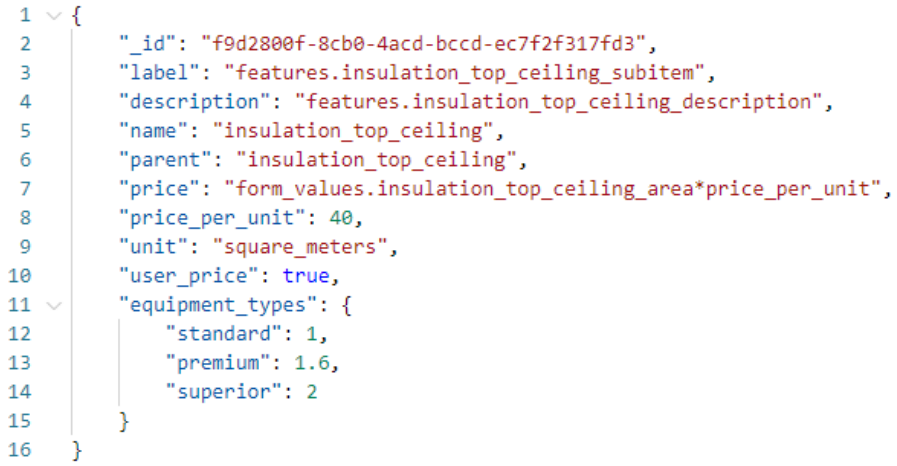



Рисунок 1.11 – Лістинг коду екземпляру сутності категорії витрат

Сутність користувача системи передбачає збереження загальної інформації про користувача, даних та токенів з мультисервісної авторизації, інформації про дату реєстрації, дату останнього входження в систему, метаданих. Користувач може авторизуватись за допомогою електронної адреси та паролю, сервісів Facebook або Google, після чого буде збережено в базі даних токен відповідного сервісу та дата, до якої токен є дійсним. Для додаткового захисту, при виконанні входу в обліковий запис, користувач може підключити запитування OTP (англ. one time password) – одноразового паролю, який запитується системою, під час виконання входу та представляє собою захищений код із 6 цифр, який генерується кожних 30 секунд.

Сутність користувача містить такі дані:

- _id – унікальний ідентифікатор користувача;
- roles – перелік ролей, які належать користувачу, такі як гість, адмін;
- login_services – дане поле сутності представляє собою об’єкт із переліком підключених сервісів для авторизації в системі.
- first_name – ім’я користувача;
- last_name – прізвище користувача;

– avatar – дане поле представляє собою посилання на аватарку користувача, яка зберігається в Google Cloud Storage – веб-службі хостингу файлів, для зберігання та доступу до файлів [11];
– last_login_at – дата останнього входу в систему;
– meta.clientIP – IP адреса користувача;
– meta.domain – домен сайту;
– created_at – дата створення облікового запису;
– last_updated_at – дата останнього оновлення облікового запису;
– email.address – електронна адреса користувача;
– email.verified – поле, яке вказує чи користувач верифікував електронну адресу;

Схему сутності користувача наведено на рисунку 1.12.

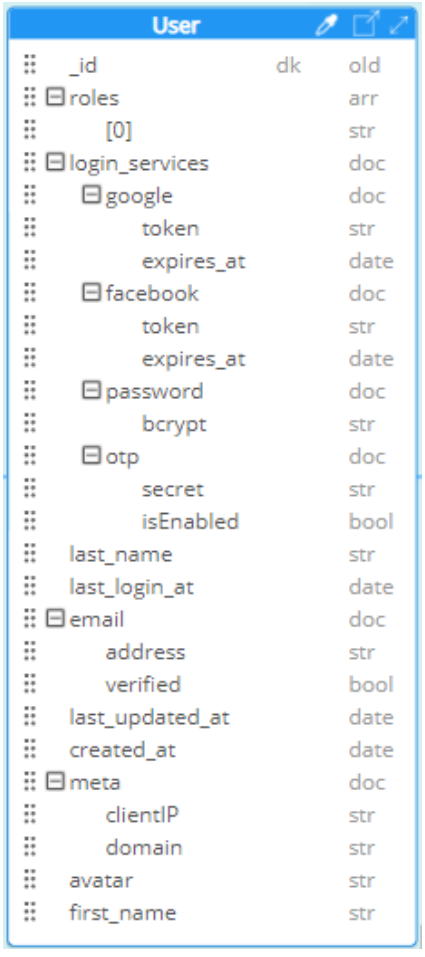



Рисунок 1.12 – Схема сутності користувача

Сутність плану проекту передбачає збереження інформації про вибрані користувачем категорії витрат. Під час створення проекту відбувається створення плану. Проект та план зв’язані між собою за допомогою зовнішнього ключа project_id. Під час зміни деталей проекту, додавання, видалення чи зміни вартості кожної з категорій, зміни типу обладнання, деталей, які містить категорія, відбувається перерахування вартості плану проекту. Сутність плану проекту містить перелік категорій витрат, кожна з яких представлена об’єктом, який містить такі дані: category_id – зовнішній ключ до колекції категорій витрат; calculated_price – вартість планованої категорії, обчисленої на основі формули; equipment_type – тип обладнання.

Сутність плану проекту містить такі дані:

– _id – унікальний ідентифікатор;
– project_id – ідентифікатор проекту, зовнішній ключ;
– price – загальна вартість проекту;
– created_at – дата створення плану;
– last_updated_at – дата останнього оновлення проекту;
– plan – поле, яке є переліком категорій витрат вибраних користувачем під час планування проекту.

Схему сутності плану проекту наведено на рисунку 1.13.

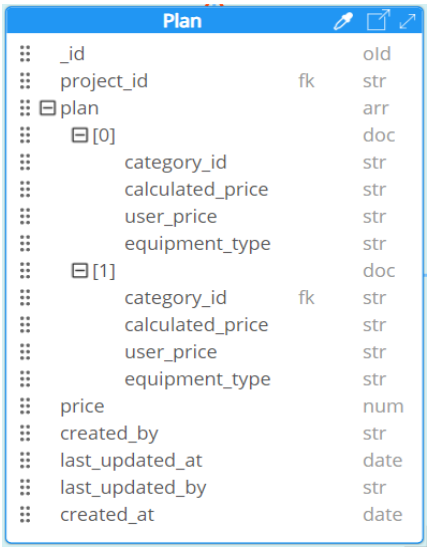



Рисунок 1.13 – Схема сутності плану проекту

Сутність платежу передбачає збереження загальної інформації про платіж, який користувач створює, на основі вибраної категорії витрат. Варто зазначити, що користувач може створювати декілька платежів, до кожної із запланованих категорій витрат, вказавши тип платежу частковий або фінальний. Дата оплати платежу зберігається для можливості групування та сортування платежів за датою їх створення, що надасть зручності в їх обліку.

Сутність платежу містить такі дані:

- project_id – ідентифікатор проекту, зовнішній ключ;
- name – назва платежу;
- notes – деталі платежу;
- amount – вартість платежу;
- date_paid – дата оплати платежу;
- category_id – ідентифікатор категорії витрат, зовнішній ключ;
- payment_type – тип платежу;
- attachments – перелік прикріплених зображень чеку про оплату;
- archived – поле, яке вказує чи було видалено платіж;
- created_by – унікальний ідентифікатор користувача, зовнішній ключ;

Схему сутності платежу наведено на рисунку 1.14.

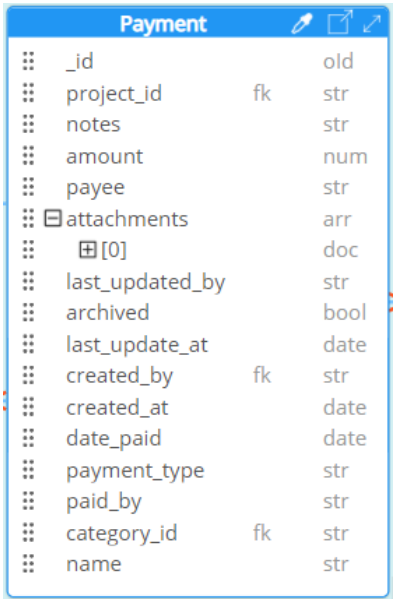



Рисунок 1.14 – Схема сутності платежу

Після проектування та опису основних сутностей системи перейдемо до проектування ER-діаграми розроблювальної веб-орієнтованої системи. Результат проектування зображено на рисунку 1.15.

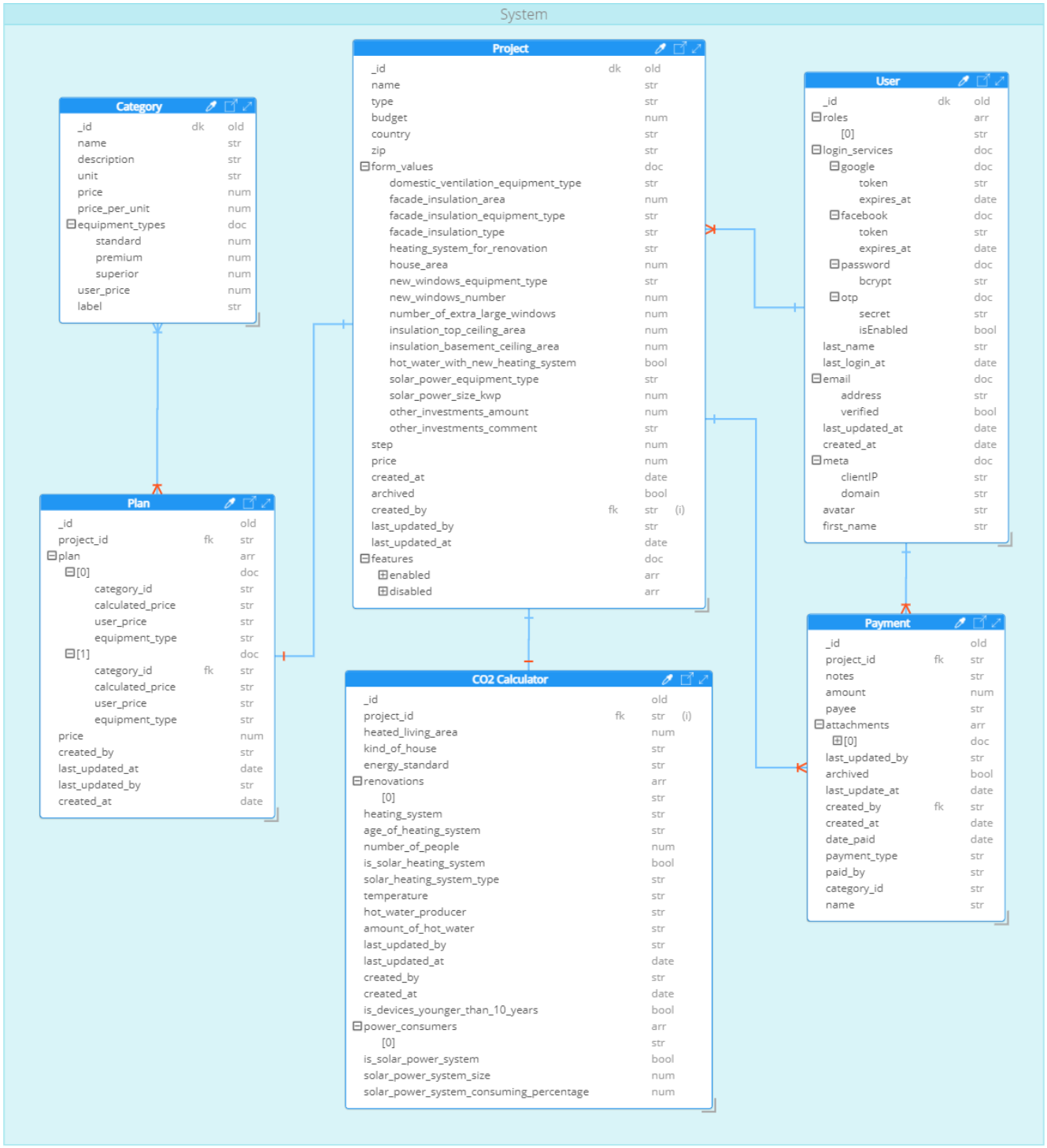



Рисунок 1.15 – Схема бази даних

Здійснивши аналіз схеми бази даних, можна ствердити, що колекції зв’язані між собою за допомогою первинних та зовнішніх ключів. Сутність $CO_2$ калькулятора зв’язана із сутністю проекту за допомогою зв’язку один до одного, тобто тільки один проект може містити одну сутність обчислень викидів вуглекислого газу, дані сутності пов’язані за допомогою зовнішнього ключа project_id. Сутність користувача пов’язана із сутністю проекту за допомогою зовнішнього ключа created_by та за допомогою зв’язку один до багатьох, це означає те, що одному користувачу може  належати декілька проектів, проте один проект належить лише одному користувачу. Сутність плану зв’язана з сутністю проекту за допомогою зовнішнього ключа project_id, використовуючи зв’язок один до одного, що  означає те, що тільки один план може належати одному проекту. Сутність категорії витрат пов’язана із сутністю плану проекту зв’язком багато до багатьох, тобто це означає, що багато планів можуть включати багато категорій витрат, дані сутності пов’язані за допомогою зовнішнього ключа ка. Сутність платежу та проекту пов’язані за допомогою зв’язку багато до одного, тобто до одному проекту може належати багато платежів, сутності пов’язані за допомогою зовнішнього ключа project_id.

### 1.3.3 Реалізація основних класів та методів

В даному розділі акцент буде наголошено на практичній реалізації розроблювальної системи.

Розглянемо основні серверні методи.

Одним з основних методів класу ProjectService, є метод для редагування проекту, програмний код цього методу наведено у рисунку 1.16.

```
async projectCalculator({ projectId, formData, meta, userId }) {
    const project = await this.projectModel.findOne({ _id: projectId }, 'type planned_project_type calculations').lean();
    if (!project) throw new UserInputError(errors.project.notFound);
    if (isEmpty(formData)) throw new Error(errors.formDataIsMissing);

    const payloadKeys = ['features', 'name', 'budget', 'country', 'zip', 'step', 'equipment_type', 'estimated_property_price'];
    const form_values = omit(formData, payloadKeys);
    const payload = { ...pick(formData, payloadKeys), form_values, meta, project_details_submitted: true };

    if (project.type === PROPERTY_VALUATION && formData.change_project_type){
      await this.changeProjectType(project, project.planned_project_type);
    }

    await this.updateProject(projectId, payload, userId);

    if(project.type === RENOVATION_HOUSE && project.co2_emissions_before) {
        const co2CalculatorFormData = get(project, 'co2_emissions_before.form_values');
        await this.co2CalculatorService.updateCO2CalculatorValues({ formData: co2CalculatorFormData, projectId });
        await this.co2CalculatorService.calculateCO2EmissionsAfter(projectId);
    }

    return this.projectModel.findOne({ _id: projectId }).lean();
}
```

Рисунок 1.16 – Програмний код реалізації методу для редагування проекту

Даний метод приймає аргумент значенням якого є об’єкт із такими властивостями: projectId – унікальний ідентифікатор проекту, formData – значення з форми, які є даними про проект, userId – унікальний ідентифікатор користувача, який виконав подання форми.

На початку методу виконується запит до бази даних на отримання документу проекту по id із колекції projects. Далі виконується перевірка чи було знайдено проект, якщо ні, то викидаємо помилку про те, що проект не було знайдено. Якщо проект із переданим id було знайдено, виконання методу продовжується. Наступним кроком є оголошення переліку даних проекту, які необхідно отримати із даних форми. Сформувавши об’єкт payload з усіма даними необхідними для редагування проекту, виконуємо виклик функції updateProject із такими аргументами: id проекту, об’єкт із редагованими даними проекту та id користувача. Після цього виконується виклик функції для оновлення розрахунків. Реалізація методу завершується пошуком та поверненням оновленого проекту за унікальним ідентифікатором проекту, Розглянемо метод редагування плану проекту на рисунку 1.17.

```
async recalculatePlan({ projectId, userId }) {
    const project = await this.projectModel.findOne({ _id: projectId }).lean();
    if (!project) {
      throw new Error(errors.project.notFound);
    }

   const existingProjectPlan = await this.projectPlanModel.findOne({ project_id: projectId }).lean();

   const plan = await this.projectPlanInit({ project, projectPlan: existingProjectPlan });

    const projectPlanId = existingProjectPlan?._id;
    if (existingProjectPlan) {
      const $set = { ...plan, last_updated_at: new Date() };
      await this.projectPlanModel.updateOne({ _id: existingProjectPlan._id }, { $set });
    } else {
     const planData = { ...plan, project_id: projectId, created_at: new Date(), created_by: userId, last_updated_at: new Date() };
       projectPlanId = await this.insertProjectPlans(planData);
     }
    await this.updatePrice({ projectId, price: plan.price, userId });
    return projectPlanId;
  }

```

Рисунок 1.17 – Програмний код методу для редагування плану проекту

Даний метод викликається щоразу, коли відбулось редагування проекту. Метод виконує перевірку чи існує план для проекту. Якщо план не було знайдено відбувається виклик методу projectPlanInit, який створює та записує план проекту в базу даних. Якщо план проекту знайдено, буде викликано метод, який виконує оновлення плану проекту в базі даних. Розглянемо метод створення плану проекту на рисунку 1.18.

```
async projectPlanInit({ project, projectPlan }) {
    const MAX_PRICE = 10000000;
    let existingPlan = projectPlan.plan || {};

    const plan = {};
    let price = 0;

    const { enabled: enabledFeatures = [] } = project.features || {};
    const planningCategories = await this.planningCategoryModel.find({ project_type: project.type, name: { $in: enabledFeatures } }).lean();

    const promisesPlanningCategories = planningCategories.map(async (category) => {
      const categoryData = { project,  category,  categoryPlan: existingPlan[category.name], overrideEquipmentType };
      const categoryPlan = await this.calculateCategory(categoryData);
      plan[category.name] = categoryPlan;
      return categoryPlan;
    });

    for await (const categoryPlan of promisesPlanningCategories) {
      if (categoryPlan && !categoryPlan.plan?.reserves_for_unexpected_costs) {
  const categoryPrice = Number.isFinite(categoryPlan.user_price) ? categoryPlan.user_price : categoryPlan.calculated_price;
      price = _.add(price, categoryPrice);
      }
    }
    const unexpectedCosts = await this.recalculateReserves({project, plan });

    price = _.ceil(price + unexpectedCosts);

    if (price > MAX_PRICE) price = MAX_PRICE;

    existingPlan = _.merge(projectPlan, { price, plan });
    return existingPlan;
  }
```

Рисунок 1.18 – Програмний код реалізації методу створення плану проекту

Даний метод приймає об’єкт плану проекту та об’єкт проекту. Далі відбувається отримання вибраних користувачем категорій для планування. Після цього відбувається розрахунок вартості кожної з категорій на основі даних введених користувачем при плануванні проекту. Після обчислення всіх категорій відбувається формування плану проекту. Наступним кроком відбувається обчислення вартості категорії для резервних витрат, яка дорівнює десяти відсоткам від загальної суми обчислених категорій. В кінці методу створений план проекту буде повернено з виклику методу.

Розглянемо основні запити на отримання даних для повернення їх на клієнтську частину. Запит GraphQL використовується для читання або отримання значень та допомагають зменшити надмірне отримання даних. На відміну від Restful API, GraphQL дозволяє користувачеві обмежувати поля, які слід отримати з сервера. Це означає менші запити та менший трафік по мережі; що в свою чергу скорочує час відповіді.

Одним із важливих запитів є отримання списку проектів користувача. Реалізацію даного запиту “projects” наведено на рисунку 1.19.

```
Query: {
    projects: async (root, params, ctx) => {
      const { userId } = ctx;
      checkAuthorization(userId);

    const projects = await this.projectModel.find({ created_by: userId,   archived: { $ne: true } }).lean();

     return projects;
  }}
```

Рисунок 1.19 – Програмний код для реалізації запиту “projects” на отримання проектів користувача

Даний програмний код виконує перевірку чи користувач авторизований. Далі виконує пошук проектів користувача за його унікальним ідентифікатором. В кінці методу дані повертаються на клієнтську частину, де в подальшому будуть відображені на сторінці списку проектів.

Для отримання даних по певному проекті, необхідні виконати запит “project”, в який передається унікальний ідентифікатор проекту. Реалізацію даного запиту наведено на рисунку 1.20.

```
Query: {
    project: async (root, params, ctx) => {
      const { id: projectId } = params;
      const { userId } = ctx;
      checkAuthorization(userId);

      const project = await this.projectModel.findOne({ projectId }).lean();
      if (!project) {
        throw new UserInputError(errors.project.notFound);
      }
      if (project.created_by !== userId) {
        throw new ForbiddenError (accessDenied);
      }
      return project;
  }}
```

Рисунок 1.20 – Програмний код для реалізації запиту “project” на отримання певного проекту користувача

Даний метод виконує перевірку чи користувач авторизований та після цього виконує пошук проекту в базі даних за ідентифікатором проекту. У випадку якщо проекту не було знайдено буде кинуто помилку про це. Якщо поле проекту created_by не буде дорівнювати ідентифікатору користувач буде кинуто помилку про відмову у запиті та завершено виконання методу. Якщо помилок не буде кинуто, тоді в кінці виконання методу буде повернено проект.

Розглянемо реалізацію клієнтської частини системи.

Популярність односторінкових веб-додатків, або SPA серед власників продуктів можна пояснити тим, що такі додатки не завантажують кожну нову сторінку, а лише витягують дані з сервера. Такий підхід зумовлює до скорочення часу відповіді та кращого досвіду користувача. Ще одним великим плюсом SPA є їх ефективність, оскільки вони дозволяють розробникам повторно використовувати код у програмі.

Інша назва архітектури SPA – товстий клієнт, тому що браузер, будучи клієнтом, містить більше логіки і виконує такі функції, як візуалізація HTML, перевірка, зміни користувальницького інтерфейсу і т.д. Односторінковий додаток – це веб-додаток або веб-сайт, який взаємодіє з користувачем, динамічно переписуючи поточну сторінку, а не завантажуючи цілі нові сторінки з сервера.

На рисунку 1.21 зображено типову архітектуру SPA з користувачем, браузером і сервером. На рисунку зображений користувач, який робить запит, і дії введення, такі як натискання кнопки, перетягування, наведення курсору миші і так далі. Отже, виділимо основний цикл кроків роботи SPA:

1. користувач вводить URL-адресу в браузері, щоб відкрити нову сторінку;
2. браузер відправляє URL-запит на сервер.
3. сервер відповідає статичними ресурсами, такими як HTML, CSS і JavaScript;
4. статичні ресурси включають код JavaScript для SPA. При завантаженні цей код виконує додаткові запити даних (запити ajax/fetch).
5. дані повертаються у форматі JSON, XML або будь-якому іншому форматі;
6. як тільки SPA отримає дані, він може відобразити відсутній HTML (блок інтерфейсу користувача на малюнку). Іншими словами, рендеринг користувальницького інтерфейсу відбувається в браузері за допомогою шаблонів, які містять дані;
7. веб-сторінка відображається користувачу, який може взаємодіяти зі сторінкою, ініціюючи нові запити від SPA до сервера, і цикл кроків 2-6 триває. На цьому етапі маршрутизація браузера може статися, якщо SPA реалізує її, тобто перехід за новою URL-адресою викличе не перезавантаження нової сторінки з сервера, а скоріше повторне завантаження SPA в браузері.

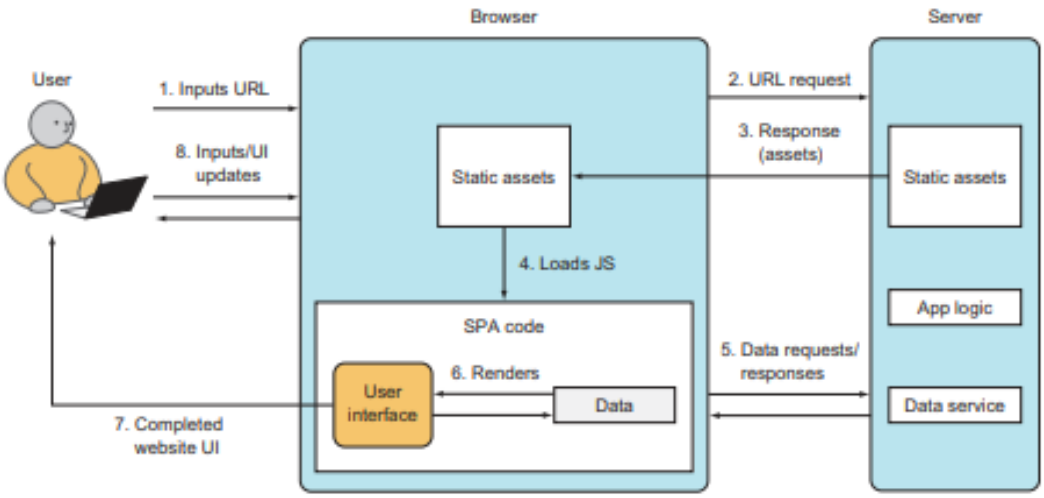
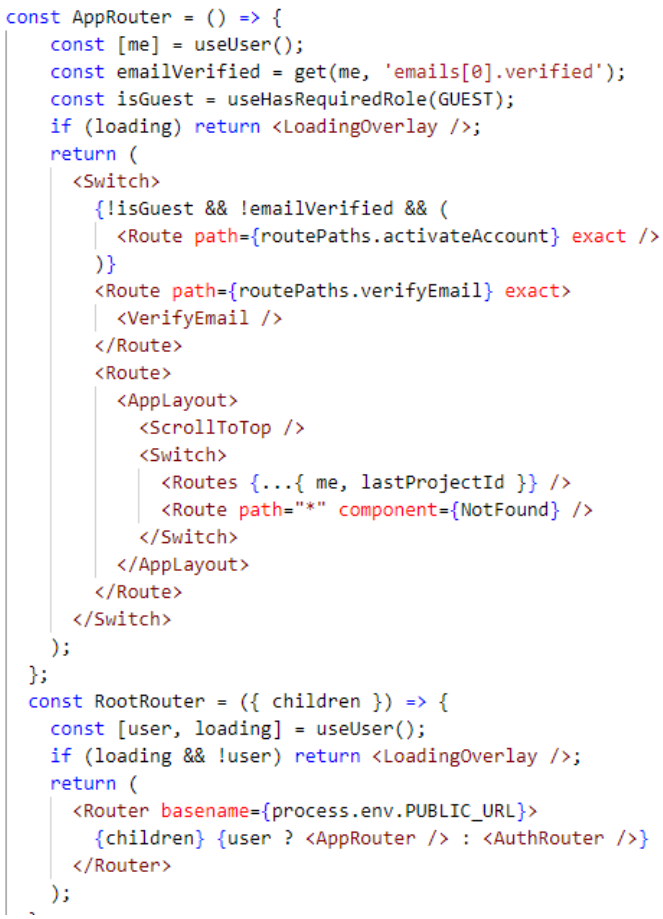



Рисунок 1.21 – Загальна структура SPA проекту на React

В підході SPA велика частина рендерингу користувальницьких інтерфейсів відбувається в браузері. Тільки дані передаються в браузер і з нього, на відміну до підходу з товстим сервером, коли весь рендеринг відбувається на сервері. При запуску застосунку, виконується код в RootRouter (див. рисунок 1.22), у якому виконується запит до бекенду на отримання інформації про користувача та перевірка авторизації в системі.

Якщо користувач не авторизований, в системі будуть доступні тільки ті сторінки, які оголошені в AuthRouter, а саме: сторінка логіну, де йому буде запропоновано здійснити вхід в систему, або можливість зареєструватись; сторінка відновлення паролю; сторінка надсилання повідомлення про проблему в системі. Після авторизації, користувача буде перенаправлено в основний AppRouter, в якому доступні такі сторінки: сторінка створення проекту; сторінка із списком створених проектів; сторінка дашборду, на якій відображається зведена інформація про проект; сторінка плану проекту; сторінка надсилання запиту на фінансування; сторінка створення та редагування платежу; сторінка порівняння планованих та фактичних витрат по кожній із планованих категорій.



```
const AppRouter = () => {
    const [me] = useUser();
    const emailVerified = get(me, 'emails[0].verified');
    const isGuest = useHasRequiredRole(GUEST);
    if (loading) return <LoadingOverlay />;
    return (
      <Switch>
        {!isGuest && !emailVerified && (
          <Route path={routePaths.activateAccount} exact />
        )}
        <Route path={routePaths.verifyEmail} exact>
          <VerifyEmail />
        </Route>
        <Route>
          <AppLayout>
            <ScrollToTop />
            <Switch>
              <Routes {...{ me, lastProjectId }} />
              <Route path="*" component={NotFound} />
            </Switch>
          </AppLayout>
        </Route>
      </Switch>
    );
  };
  const RootRouter = ({ children }) => {
    const [user, loading] = useUser();
    if (loading && !user) return <LoadingOverlay />;
    return (
      <Router basename={process.env.PUBLIC_URL}>
        {children} {user ? <AppRouter /> : <AuthRouter />}
      </Router>
    );
```

Рисунок 1.22 – Програмний код реалізації роутингу в розроблювальній системі

Управління даними в програмах є складним завданням. Більшість програм вимагає: окремі клієнти інтерфейсу для кількох платформ, кожна з яких має різні вимоги до даних; бекенд, який обслуговує дані клієнтам з кількох джерел; складне керування станом і кеш-пам'яттю як для інтерфейсу, та бекенду; Використовуючи GraphQL і Apollo, можна значно зменшити ці проблеми. Декларативна модель GraphQL допомагає створити послідовний, передбачуваний API, який можна використовувати для всіх клієнтів.

Розролювальна веб-орієнтована система для керування станом застосунку використовує бібліотеку Apollo Client, яка дозволяє керувати локальними та віддаленими даними за допомогою GraphQL.На рисунку 1.22 наведено реалізацію коду піключення бібліотеки Apollo.

```
import React from 'react';
import { ApolloProvider } from '@apollo/client';
import { LocaleContextProvider } from 'contexts/LocaleContext';
import IntlProviderWrapper from 'contexts/IntlProviderWrapper';
import { AppContextProvider } from 'contexts/AppContext';
import RootRouter from './router/RootRouter';
import apolloClient from './graphql/apollo';

const App = () => (
  <ApolloProvider client={apolloClient}>
    <LocaleContextProvider>
      <IntlProviderWrapper>
        <AppContextProvider>
          <RootRouter>
            <TrackPage />
          </RootRouter>
        </AppContextProvider>
      </IntlProviderWrapper>
    </LocaleContextProvider>
    <ToastContainer />
  </ApolloProvider>
);
```

Рисунок 1.22 – Програмний код підключення Apollo провайдера для реалізації керування станом застосунку

Під час додавання, видалення та міграції внутрішніх сховищ даних цей API не змінюється з точки зору клієнта. Незважаючи на багато інших переваг, найбільшою перевагою GraphQL є досвід розробника, який він надає. Легко додавати нові типи та поля до API, і так само легко для клієнтів різних типів почати використовувати ці поля. Це допоможе швидко проектувати, розробляти та впроваджувати функції.

Завдяки GraphQL в сильній типізації та вбудованій підтримці для самоаналізу, інструменти розробника для GraphQL є надзвичайно потужними. Ці інструменти дозволяють виконувати такі дії: досліджувати повну структуру схеми разом із рядками документів; створюти нові операції з перевіркою в реальному часі та автозавершенням; зареєструвати свою схему в службі керування, яка відстежує та перевіряє зміни. У поєднанні з платформою Apollo складні речі, такі як кешування, нормалізація даних та оптимістична візуалізація інтерфейсу користувача, також стають простими.

# 2 ВИКОРИСТАННЯ ПРОГРАМНОЇ СИСТЕМИ

## 2.1 Розгортання програмної системи та системні вимоги

Для розгортання розроблювальної клієнтської частини було обрано платформу Netlify. Netlify – це інтегрована платформа для створення, розгортання та масштабування сучасних веб-додатків. Об'єднуючи екосистему фреймворків Javascript, інструментів розробника та API в спрощені робочі процеси, Netlify допомагає командам розробників швидше поставляти та забезпечувати найкращий веб-досвід.

Використання платформи Netlify надає такі переваги:

1. Атомарні розгортання з миттєвою публікацією та відкатом. Це надзвичайно корисна властивість, яка буде невід'ємний інструментом який включає такі речі, як «розгортання», «запуск» або «перехід» в плані проекту.

Кожна успішна збірка на Netlify призводить до щойно розгорнутого екземпляру сайту. Публікація на периферійних вузлах CDN Netlify і такі речі, як анулювання кешу, відбуваються автоматично і майже миттєво.

Розгортання є незмінними. Тобто кожне розгортання призводить до версії сайту, яка ніколи не змінюється. Подальші оновлення створюють нові екземпляри сайту на заміну попередніх версій. Це означає, що надається можливість можете будь-коли повернутися до будь-якої попередньої збірки одним натисканням кнопки на консолі адміністратора або через API.

2. Повідомлення та постійні посилання. Netlify дозволяє налаштувати сповіщення на основі різних подій розгортання. Платформа надає можливість визначити, хто отримуватиме сповіщення про такі події, як початок розгортання, або коли розгортання успішно, невдало, заблоковано чи розблоковано. Наступною можливістю є налаштування сповіщень на адреси електронної пошти або на канал Slack. Що робить ці сповіщення ще більш корисними, так це те, що вони містять постійне посилання на розгортання, про яке йдеться. Всі розгортання є незмінними

і зберігаються постійно. Це означає, що кожен може мати власну URL-адресу, до якої є можливість отримати доступ, щоб побачити це розгортання.

Наявність постійних посилань для кожного розгортання величезна. Це миттєво відкриває можливість поділитися станом будь-якої версії збірки з командою тестування, клієнтом. І цей миттєвий доступ надається прямо із кожного сповіщення.

3. Розгортання гілок та субдоменів. Це дійсно зручно мати можливість розгортати гілки, крім master. Створення нових функцій у гілках функцій, а потім їх тестування та перевірку у реальному середовищі хостингу є неймовірно потужним.

Netlify дає деякий контроль над тим, як відбувається розгортання. Є можливість вибрати, чи розгортати лише свою виробничу гілку, усі гілки чи деякі іменовані гілки. Створивши репозиторій в Github, потрібно запушити проект в створений репозиторій. Після цього виконаємо розгортання системи за допомогою інструментів Netlify UI. Наступним кроком необхідно підключити обліковий запис Netlify до облікового запису Github. Після авторизації Netlify Auth відбувається перенаправлення на сторінку профілю. Далі потрібно створити новий сайт з Git. На сторінці створення нового сайту буде запропоновано авторизувати Netlify на доступ до сховищ Github. Є можливість надати Netlify доступ до всього облікового запису Github, або надати Netlify доступ до певного репозиторію, натиснувши кнопку «Налаштувати Netlify на GitHub» внизу сторінки.

Після вибору репозиторію, і вибору гілки розгортання з кроку вище, вказуємо основні параметри збірки, необхідні для розгортання сайту. Хоча ця сторінка попередньо заповнена кроком складання, потрібно переконатися, що вона відповідає команді, яка створює програму. Наступним кроком було налаштовано змінні середовища.

Після того, як попередньо переглянувши розгорнутий додаток React і якщо він виглядає добре, тепер необхідно налаштувати відповідний домен для обслуговування сайту. На вкладці “Deploys” є можливість бачити деталі розгортання сайту. Саме тут потрібно зареєструвати субдомен, який хочемо

використовувати для нашої системи. Якщо хочемо використовувати попередньо зареєстрований домен, потрібно буде підтвердити домен.

Після успішної перевірки можемо отримати доступ до нашого розгорнутого сайту у налаштованому домені.

Для розгортання серверної частини системи було обрано платформу Heroku. Heroku – хмарна PaaS-платформа, що підтримує такі мови програмування як Java, Node.js, Scala, Python [12].

Підтримка Heroku Node.js застосовуватиметься лише тоді, коли додаток має package.json файл у кореневому каталозі. Файл визначає залежно, які повинні бути встановлені разом з додатком. Щоб створити package.json файл потрібно запустити команду npm init в кореневому каталозі. Ця команда допоможе створити package.json файл. Необхідно переконатися, що пакети встановлюються не на системному рівні. Версія Node.js, яка буде використовуватися для запуску програми на Heroku, також має бути визначена у package.json файлі. Потрібно завжди вказувати версію Node.js, яка відповідає тій версії, на якій відбувається розробка та тестування.

Щоб визначити, як запустити сервер, Heroku спочатку шукає файл Procfile. Команда у веб-процесі повинна бути прив'язана до номера порту, зазначеного у PORT змінній середовища. Якщо це не так, Heroku контейнер не запуститься. Після того, потрібно виконати фіксування змін в git та розгорнути сервер на Heroku. Необхідно запустити командуgit push heroku master для розгортання серверу.

## 2.2 Опис типових схем використання системи

При першому вході в систему користувачу пропонується вибрати тип проекту. Якщо користувач вже зареєстрований в системі він може виконати вхід в систему за допомогою сервісів Google, Facebook або за допомогою електронної адреси та паролю. Сторінку логіну зображено на рисунку 2.1.

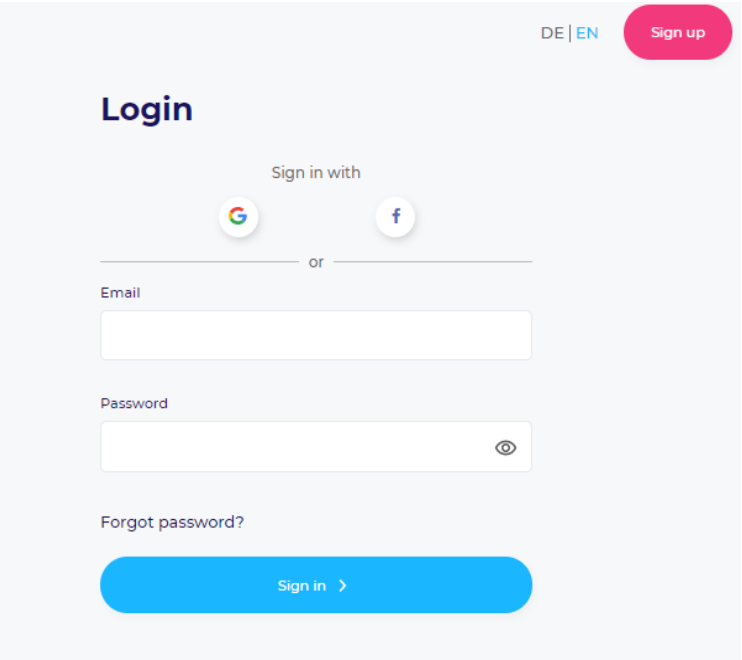



Рисунок 2.1 – Сторінка логіну системи

У налаштуваннях користувач має можливість обрати бажану мову інтерфейсу (див. рис.2.2).

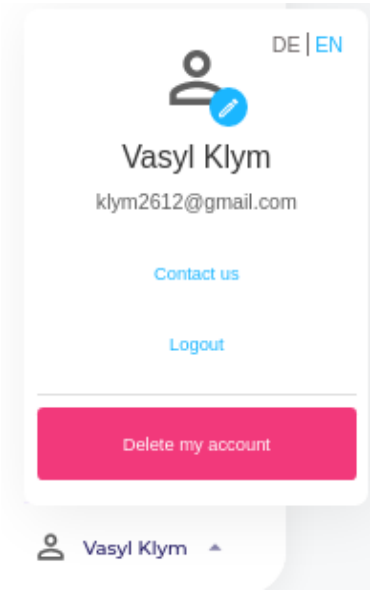



Рисунок 2.2 – Вибір вибір мови інтерфесу

Після вибору типу проекту, відбувається перенаправлення користувача на покрокову форму створення проекту. Першим кроком створення проекту, являється форма з калькулятором викидів вуглекислого газу, який виробляється

житловими будинками. На даній формі користувач повинен заповнити дані форми такі як: опалювальна житлова площа; тип будинку: односімейний будинок, бунгало, таунгауз, зблокований будинок – житловий будинок, побудований з двох частин, які часто є симетричними; енергетичний стандарт будинку; чи були проведені ремонтні роботи такі як: заміна вікон; утеплення даху, або стелі; утеплення стелі нижнього поверху; утеплення фасаду; контрольована вентиляція житлового приміщення. Перелічені дані форми зображено на рисунку 2.3.

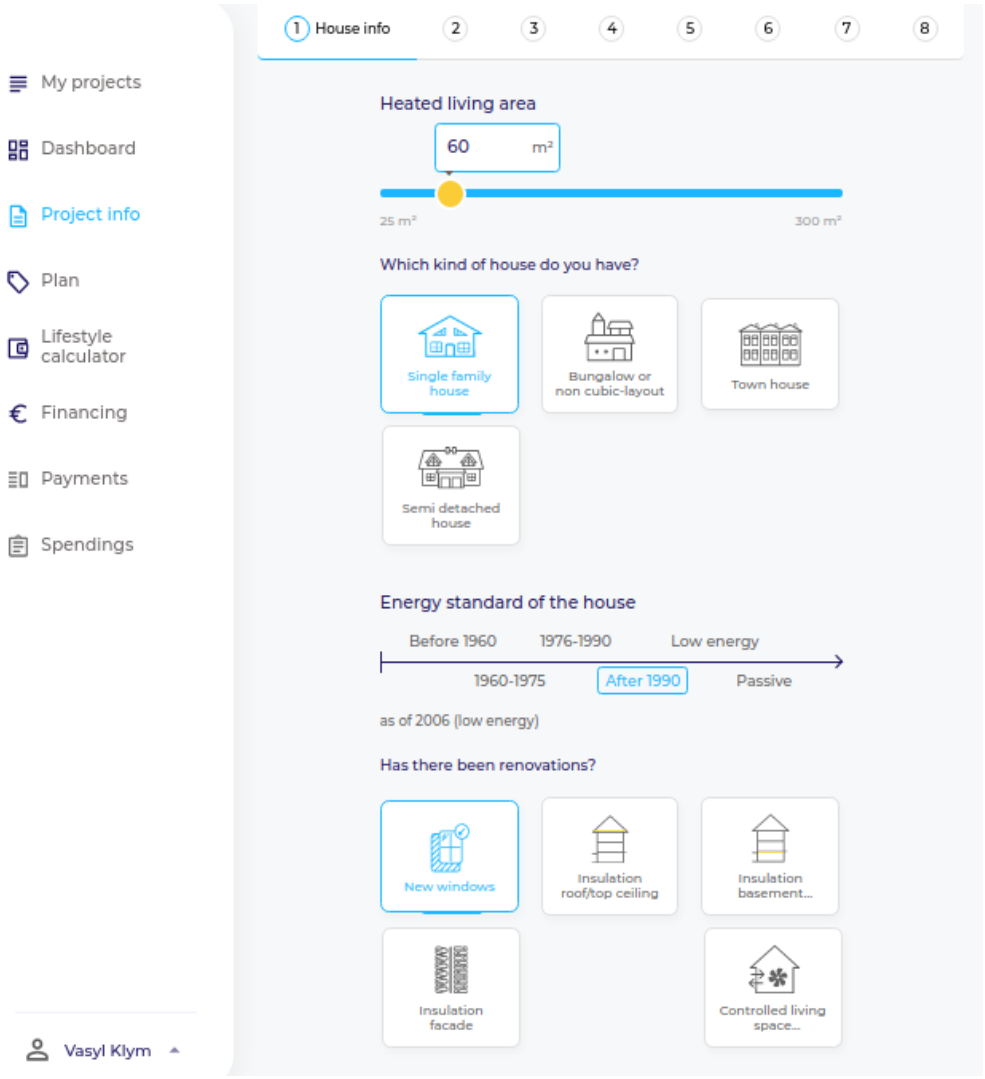



Рисунок 2.3 – Сторінка з формою калькулятора викидів вуглекислого газу житловим будинком

Наступними даними, які необхідні для обчислення енергоефективності будинку є кількість людей, яка проживає живе в будинку та тип опалювальної

системи: тепловий насос, використання біомаси в якості палива у системах централізованого теплопостачання, стандартне центральне опалення, деревна тріска паливні гранули, вугілля, електричне опалення, природній газ. Перелічені дані форми зображено на рисунку 2.4.

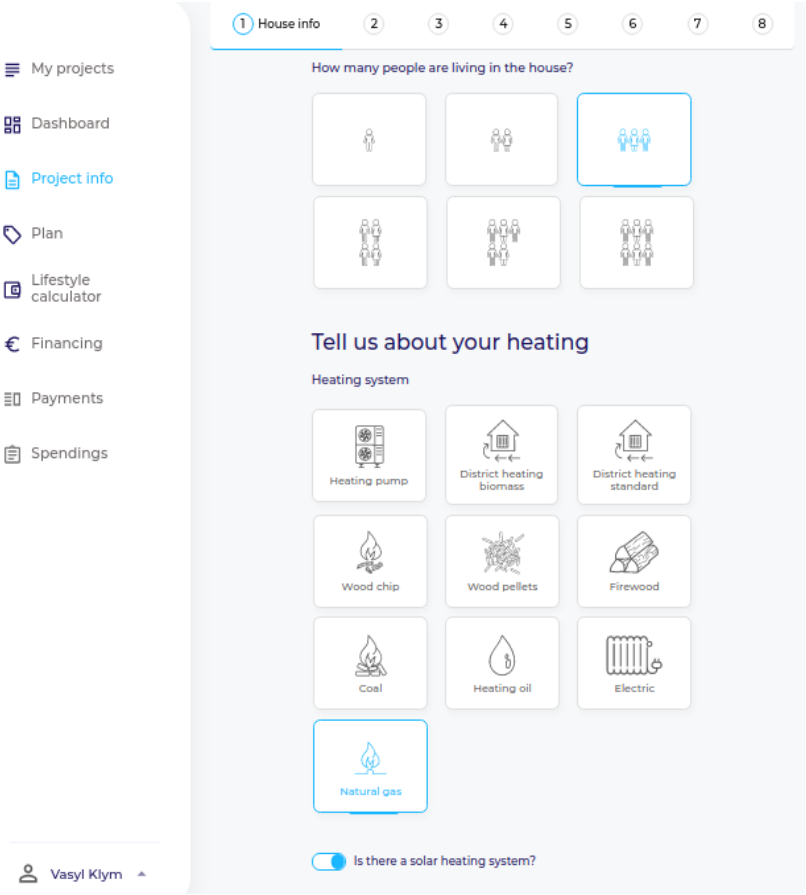



Рисунок 2.4 – Сторінка з формою калькулятора викидів вуглекислого газу житловим будинком

Також для обчислення енергоефективності будинку нам потрібно дізнатись який вік опалювальної системи: більше ніж 20 років, між 10 та 20 рків, нова технологія; яка температура в будинку: менше 21 °C, між 21 та 23 °C, більше 23 °C; який виробник гарячої води: з системою опалення, електричний котел, газовий нагрівач, тепловий насос; який необхідний рівень гарячої води: низький (тільки душ), середній (50л. в день), високий (тільки ванна); вибір у формі перемикача чи є більшість пристроїв, як-от кухонне та пральне обладнання, плита, духовка

молодше 10 років; вибрати незвичні споживачі енергії наведені на формі; чи є сонячна енергетична система і якщо так, то який її розмір та наскільки високе самоспоживання у відсотках. Перелічені дані форми зображено на рисунку 2.5.

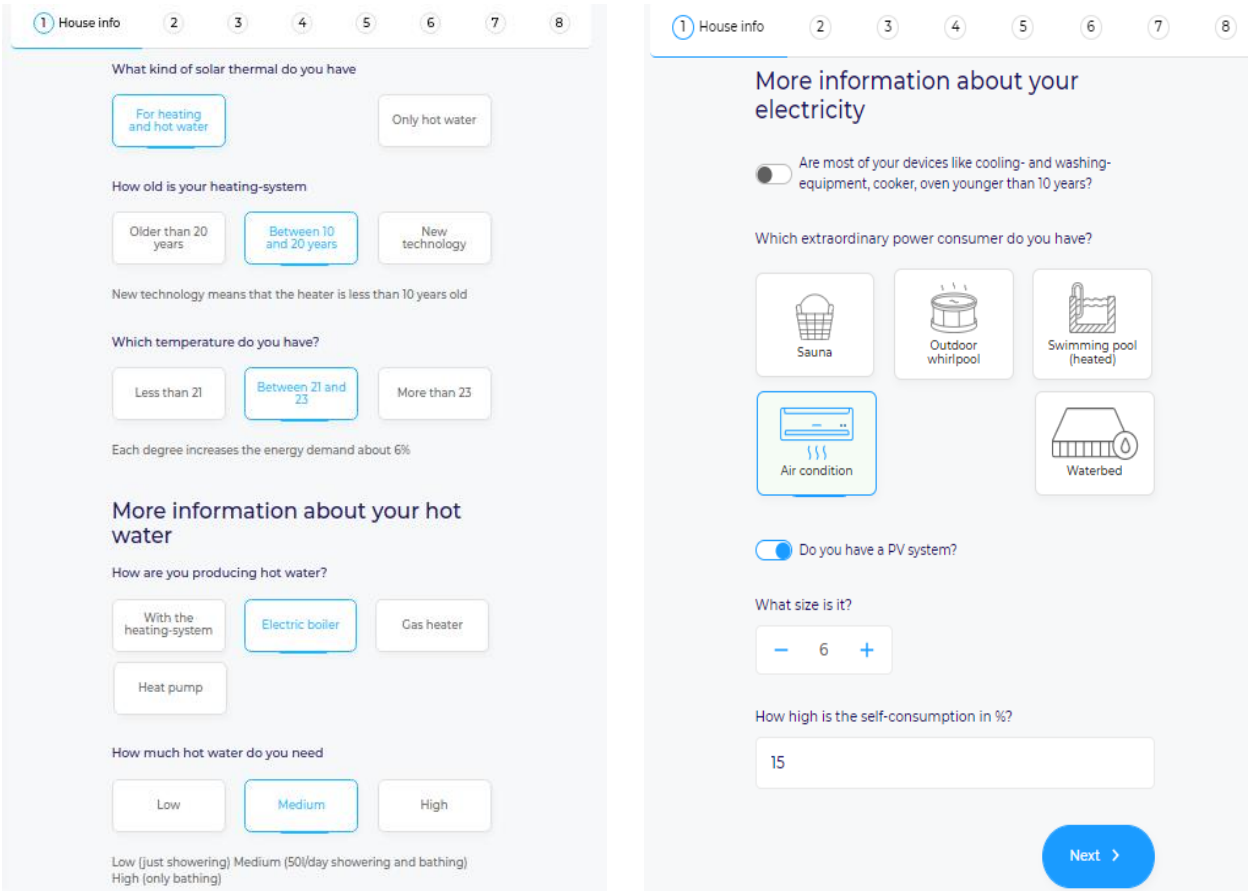



Рисунок 2.5 – Сторінка з формою калькулятора викидів вуглекислого газу житловим будинком

Після того як користувач пройшов перший крок калькулятора викидів вуглекислого газу, який виробляється житловими будинками, система перенаправляє його на наступний крок. Другим кроком являється результат обчислення системою енергоефективності будинку (див. рис. 2.6). На даному кроці відображається поточний енергетичний баланс будинку, $CO_2$-податок, енергетичні показники опалення, гарячої води та електрики для будинку без ремонтних заходів. потенціал збереження $CO_2$, скорочення витрат на електроенергію та споживання енергії після проведення повного ремонтування та встановлення сонячної

енергетичної системи (5kwp). На даному кроці система розрахувала загальну річну потребу в енергію, баланс парникових газів на рік, відповідність збережених викидів $CO_2$ до кількості кілометрів, які може проїхати автомобіль, кількості авіаперельотів та кількості збережених дерев.

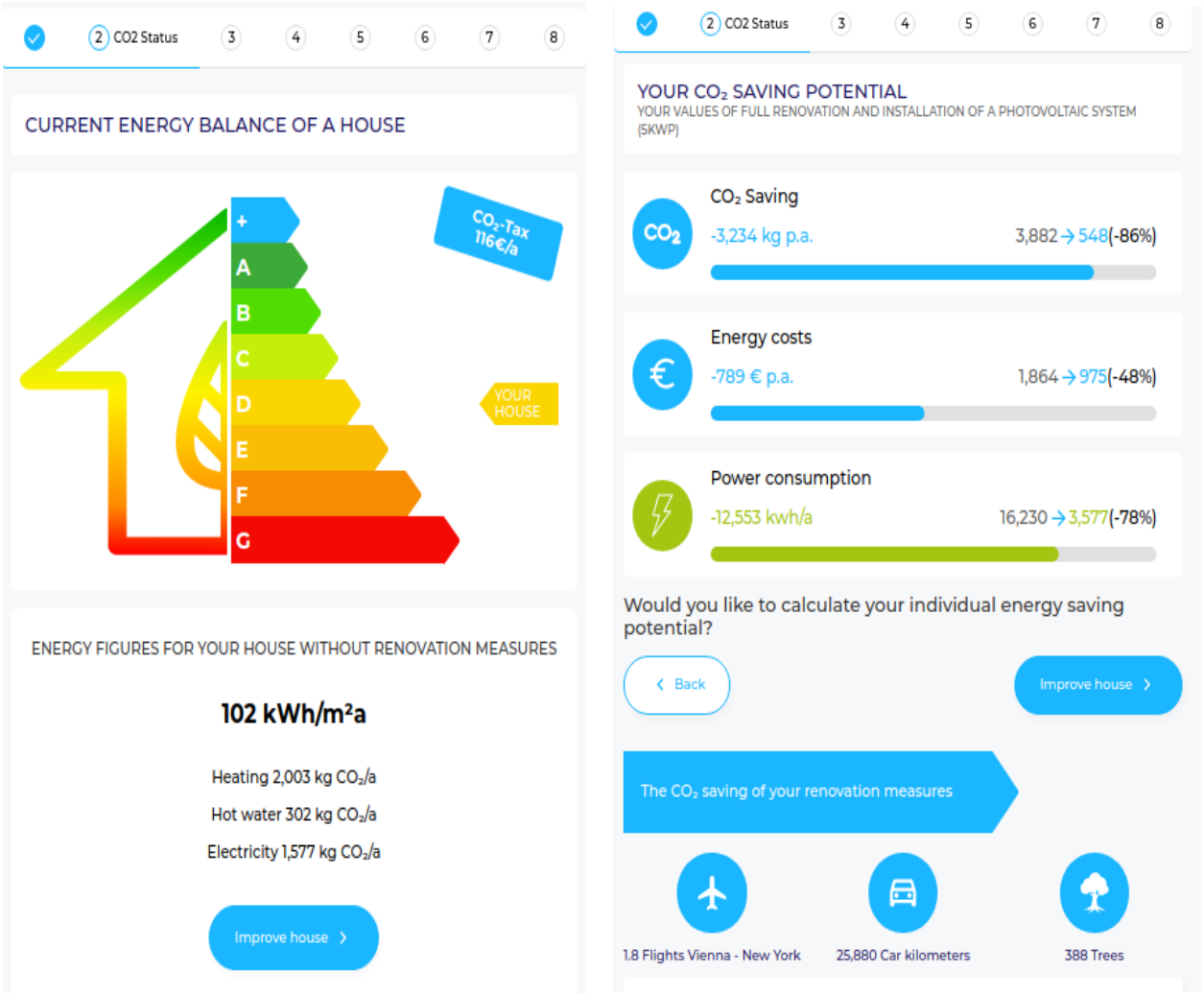



Рисунок 2.6 – Результат обчислення енергоефективності будинку

На 3 кроці сторінки планування деталей проекту (див. рис. 2.7) система пропонує заповнити таку інфомацію про проект: назва проекту, поштовий індекс, корисну житлову площу, кількість поверхів. На 4 кроці даної сторінки доступні енергетичні категорії витрат.

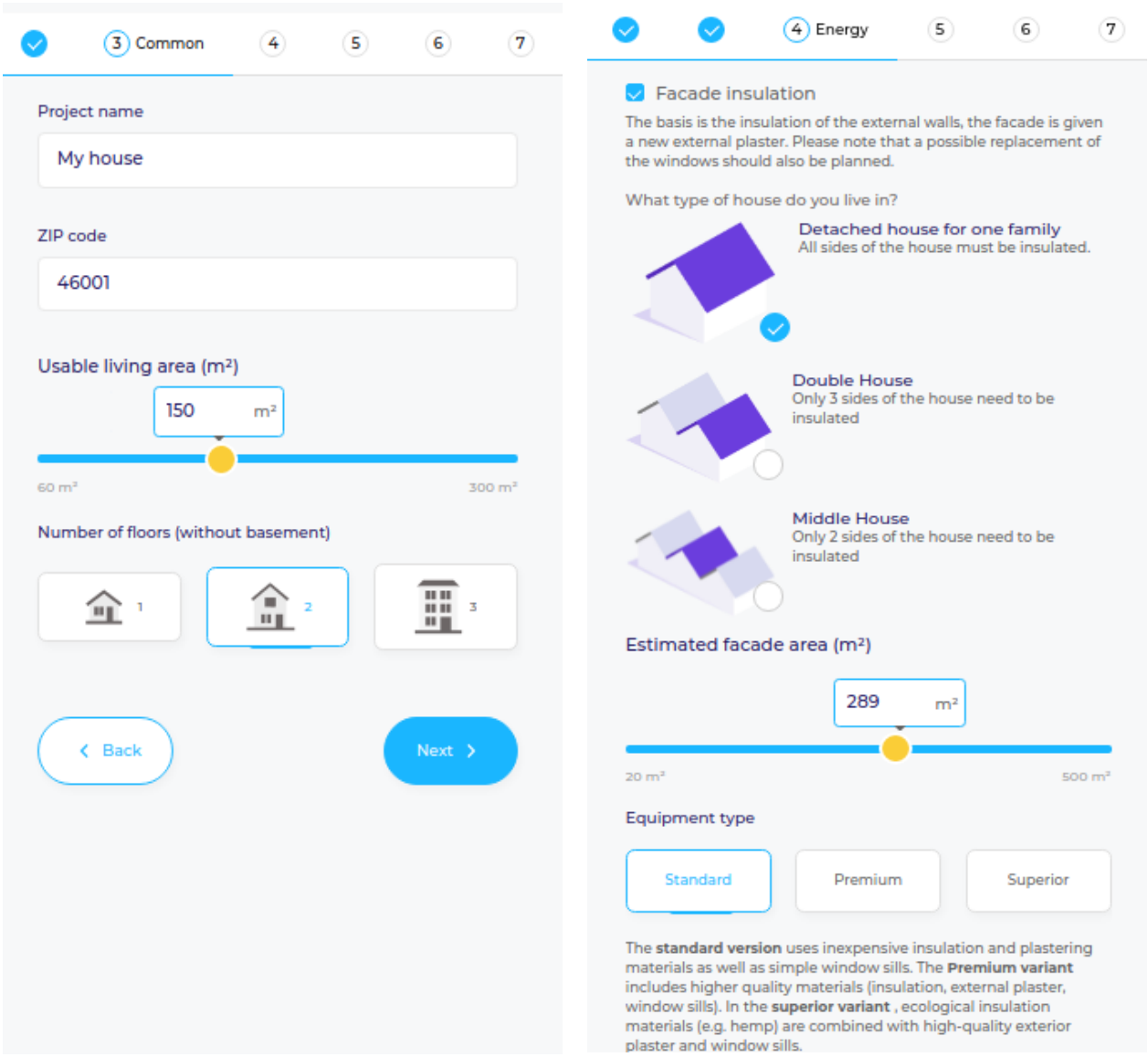



Рисунок 2.7 – Сторінка планування деталей проекту

Вибравши категорію «Утеплення фасаду», ми можемо побачити деталі вибраної категорії, де потрібно вибрати тип будинку, для того щоб дізнатись скільки сторін будинку потребують утеплення. Ввести оціночну площу фасаду, та вибрати один із типів обладнання та матеріалів для виконання робіи даного типу. У варіанті «Standard» використовуються недорогі утеплювачі і штукатурні матеріали, а також прості підвіконня. Варіант «Premium» включає в себе якісніші матеріали. У кращому варіанті «Superior» екологічні ізоляційні матеріали.

На 4 кроці сторінки планування деталей проекту (див. рис. 2.8) система надає можливість вибрати бажані енергетичні категорії витрат, такі як: утеплення стелі

верхнього поверху, утеплення стелі нижнього поверху для мінімізації втрати тепла, оновлення системи опалення, встановлення сонячної енергетичної системи, її розмір та тип обладнання, встановлення вентиляції житлових приміщень, яка забезпечує автоматичний та енергоефективний повітрообмін через припливні та вихідні отвори, встановлення настінної коробки для зарядки електромобілів, енергетичний сертифікат, який видається сертифікованими органами і необхідний для багатьох заявок на фінансування.

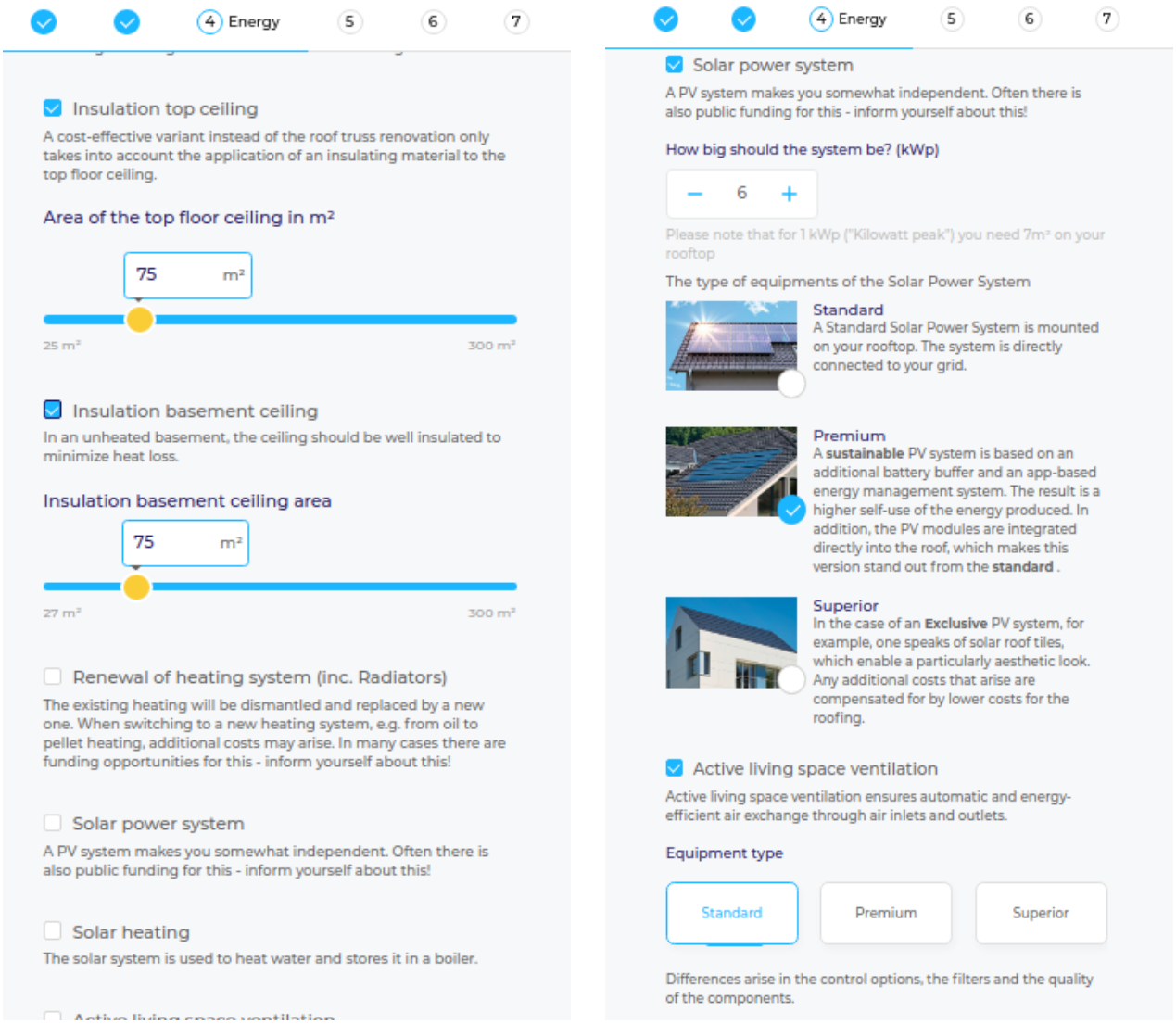



Рисунок 2.8 – Сторінка планування деталей проекту

На сторінці категорій витрат плану проекту (див. рис. 2.9) користувач має можливість переглянути вартість кожної із планованих категорій, переглянути

вартість категорії відповідно до різних типів обладнання (standard, premium, superior), а також змінити розраховану ціну систему на власну. Варто зауважити, що після кожної зміни вартості категорії витрат відбувається переобчислення вартості проекту.

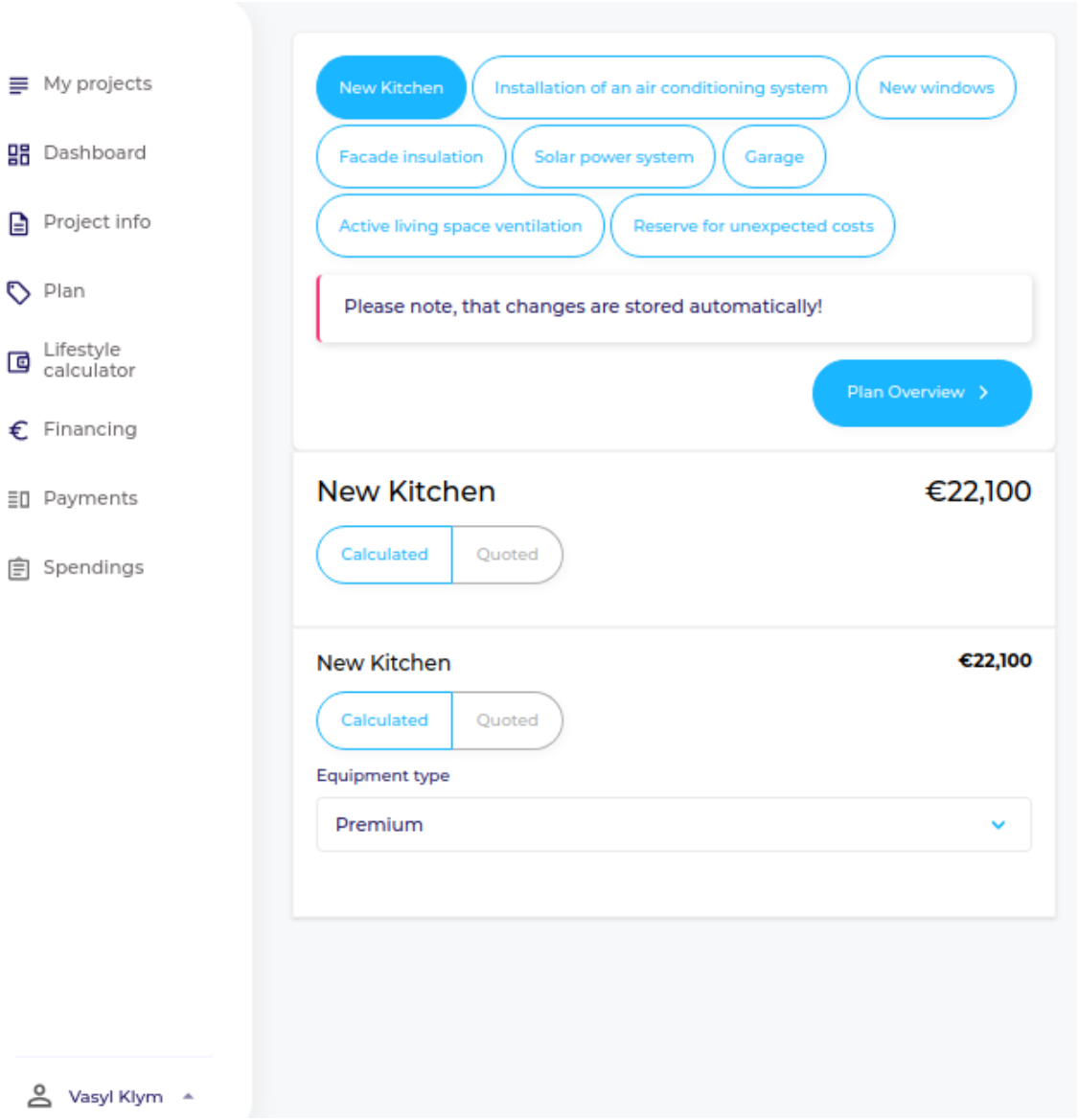



Рисунок 2.9 – Сторінка категорій плану проекту

На сторінці плану проекту (див. рис. 2.10) користувач має можливість у зручному вигляді побачити загальну вартість проекту, вартість кожної із вибраних категорій витрат, діграму, яка візуально демонструє відсоток вартості кожної з категорій до загальної вартості проекту. На даній сторінці користувач може

виконати завантаження плану проекту у форматі PDF, а також перейти до наступного кроку або повернутись до попереднього.

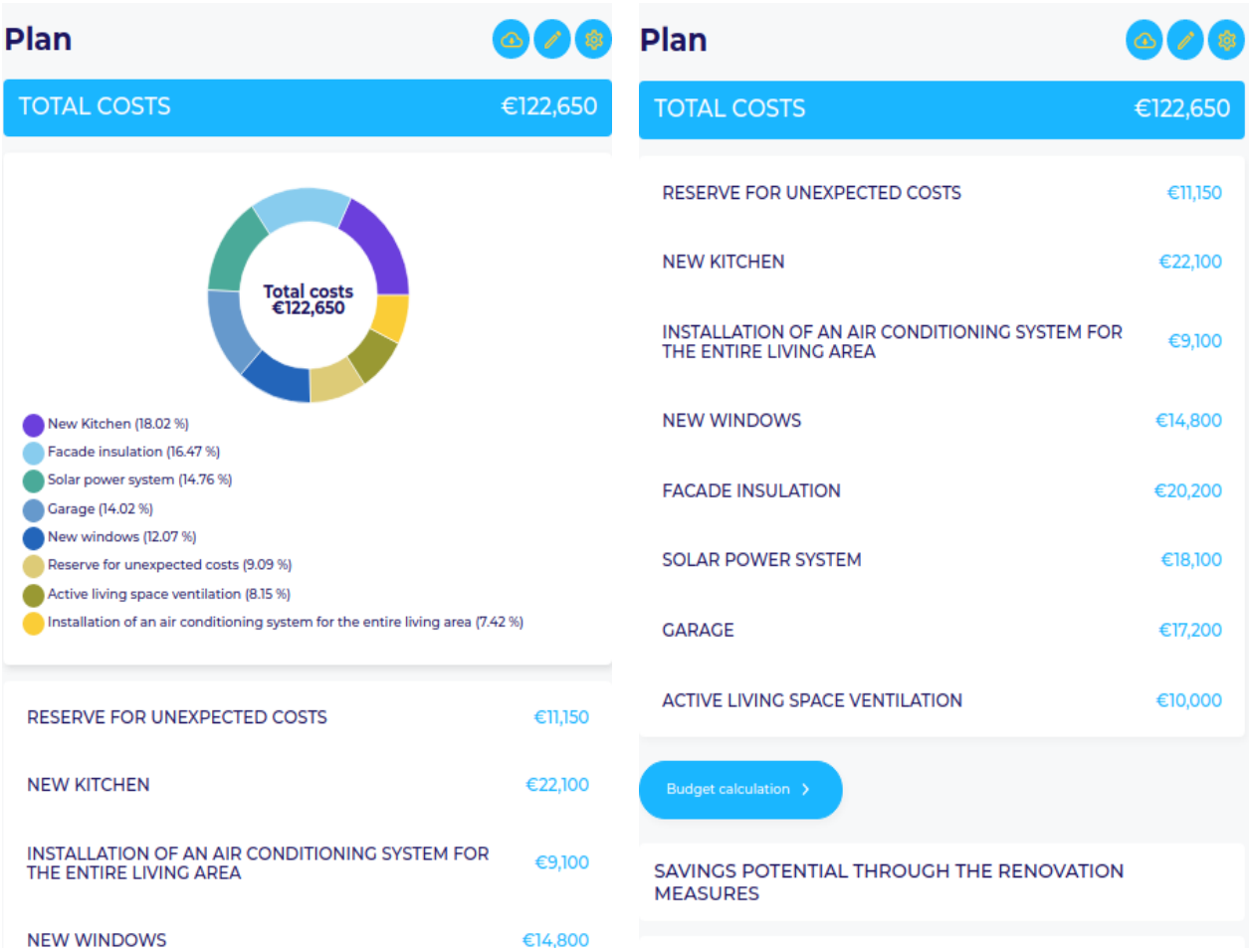



Рисунок 2.10 – Сторінка плану проекту

На сторінці “Dashboard” зображено назву проекту, загальну вартість проекту, бюджет, щомісячні доступні виплати, сума кредиту на проект, щомісячна ставка, а також перелік кроків, які користувач проходить під час планування проекту, такі як: планування будинку, планування бюджету способу життя, налаштування бюджетого плану, планування фінансування, запит на особисте консультування стосовно банківського фінансування. Сторінку “Dashboard” зображено на рисунку 2.11.

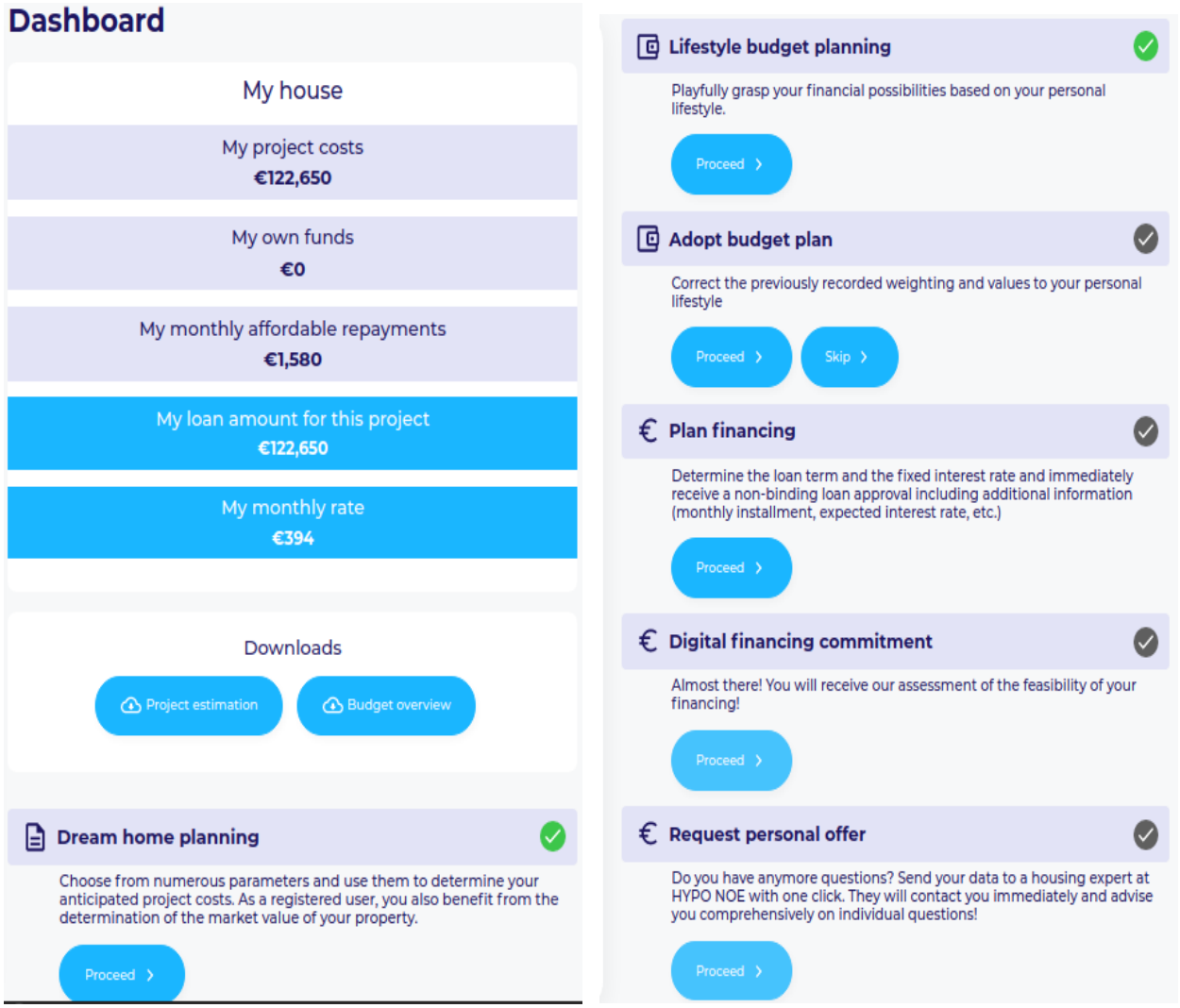



Рисунок 2.11 – Сторінка прогресу планування проекту

На сторінці фінансування зображено загальну вартість проекту, користувачу надається можливість ввести свій бюджет, на основі цих даних розраховується скільки коштів користувачу не вистачає для реалізації проекту.

У випадку недостачі коштів, користувач має можливість пройти форму, зображену на рисунку 2.12, вибравши бажаний термін позики в роках та тип процентних умов та основі обчислених значень виконати запис на особисте консультування стосовно банківського фінансування.

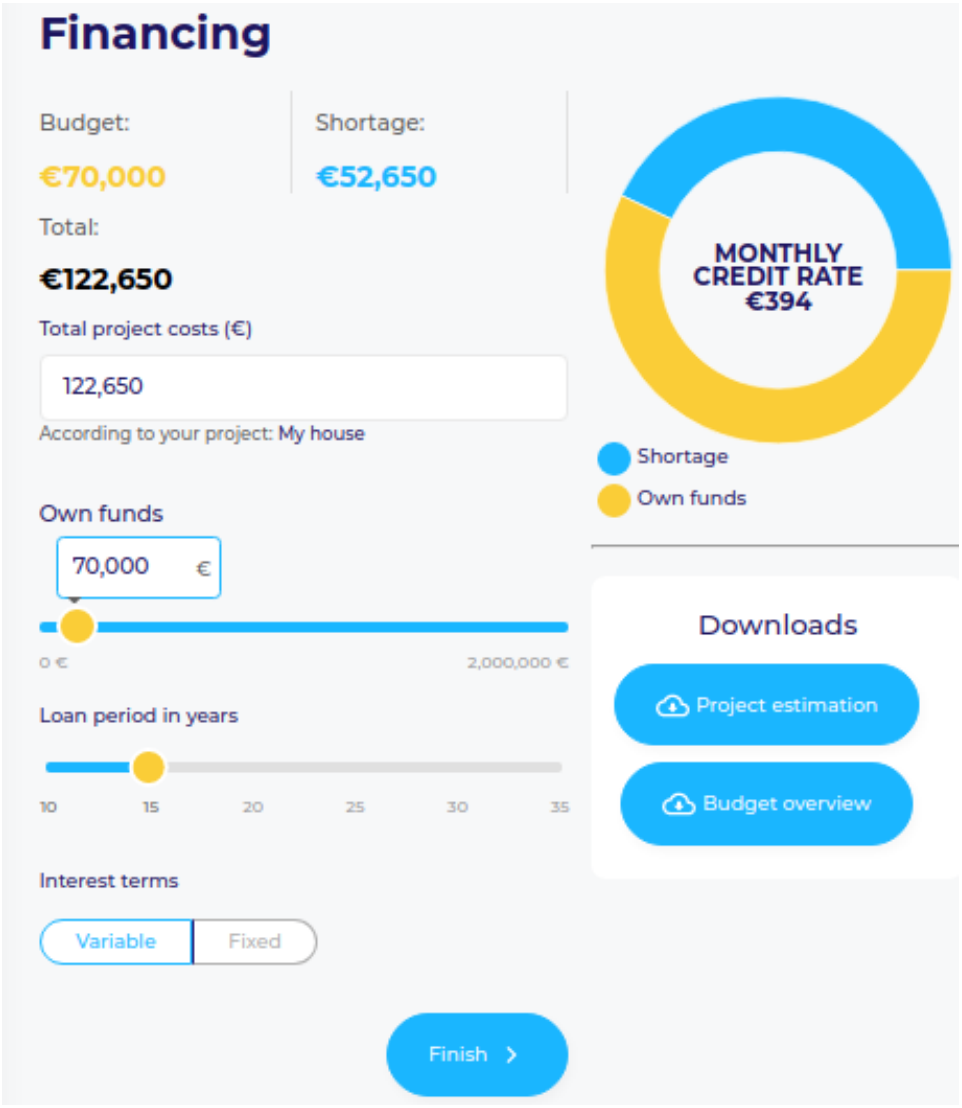



Рисунок 2.12 – Сторінка фінансування

У розроблювальній системі планується інтегрування банку. У випадку якщо у користувача недостатньо коштів на реалізацію спроектованого проекту, він має можливість виконати запит на особисте консультування стосовно фінансування, заповнивши форму із такими даними: ім’я, прізвище, електронна адреса, номер телефону, зручний час для отримання консультування, а також деталі, які можуть бути корисними для фінансової пропозиції. Реалізована форма зображена на рисунку 2.13.

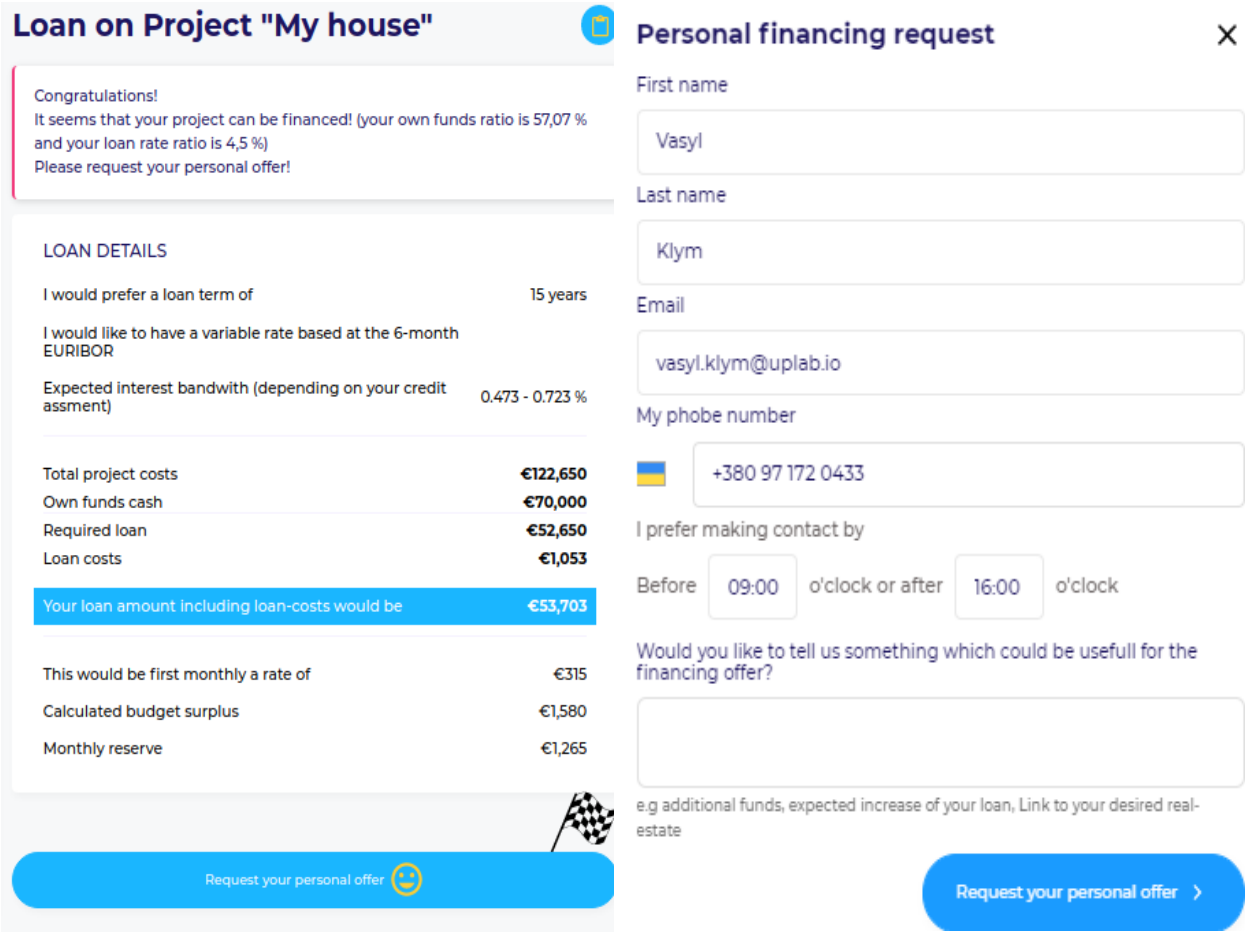



Рисунок 2.13 – Сторінка із формою запиту на особисте фінансування

Виконавши попередні кроки, користувач може розпочати створення платежів, натиснувши на пункт меню “Payments”. Для додавання нового платежу, потрібно натиснути кнопку “Record payment”, після чого буде відкрито модальне вікно з формою див. рис. 2.14, де користувачу буде необхідно заповнити такі поля: назва платежу, опис, тип сервісу, назву компанії, суму, дату платежу, вибрати платника, додати вкладення.

Після натискання кнопки “Save payment”, платіж буде додано. Після цього матеме можливість його переглянути, відредагувати або видалити.

Інформацію із вище описаної та заповненої форми зображено на рисунку 2.14, платіж в категорію “Solar power system” на суму 13000 євро.

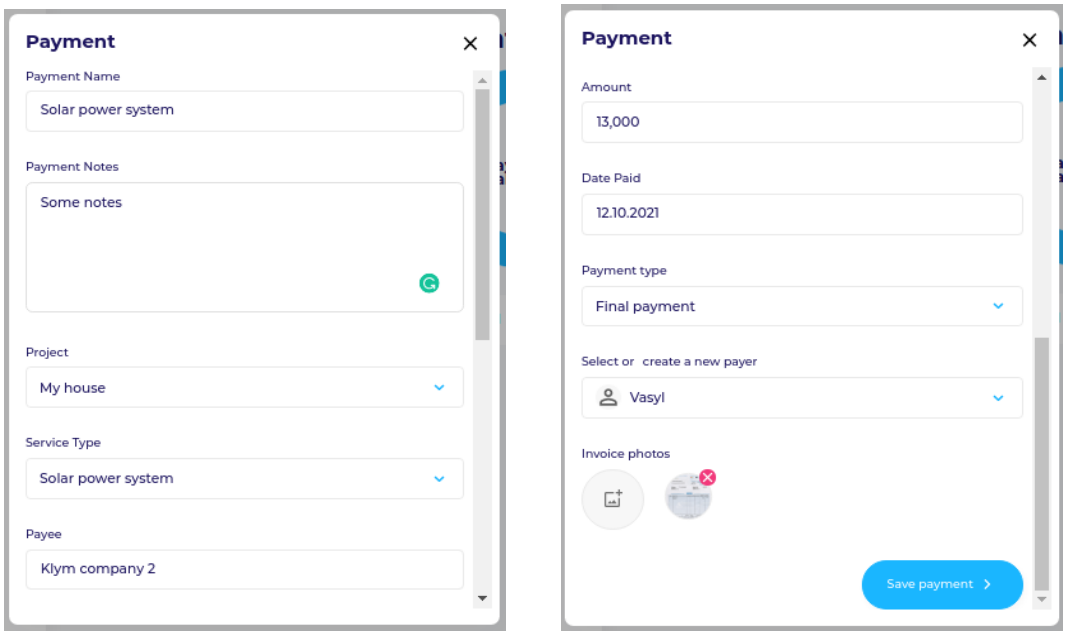



Рисунок 2.14 – Форма додавання платежу

На сторінці платежів (див. рис. 2.15) користувач має можливість переглянути узагальнену інформацію про записані платежів, зробити пошук по ключових словах, здійснити фільтрування платежів за проектом, категорією платежу та платниками. Проаналізувати діаграму, яка показує загальну суму платежів витрачену кожним платником.

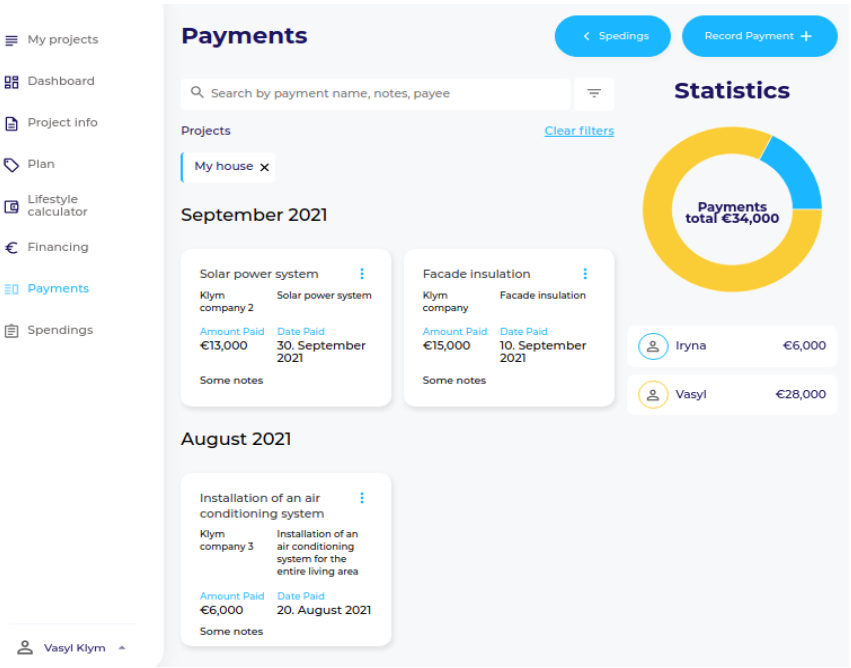



Рисунок 2.15 – Сторінка створених платежів

Перейшовши на сторінку планування витрат, користувач матиме змогу побачити заплановану вартість витрат по кожній з категорій та фактичні здійсненні витрати. Сторінку планування витрат зображено на рисунку 2.16.

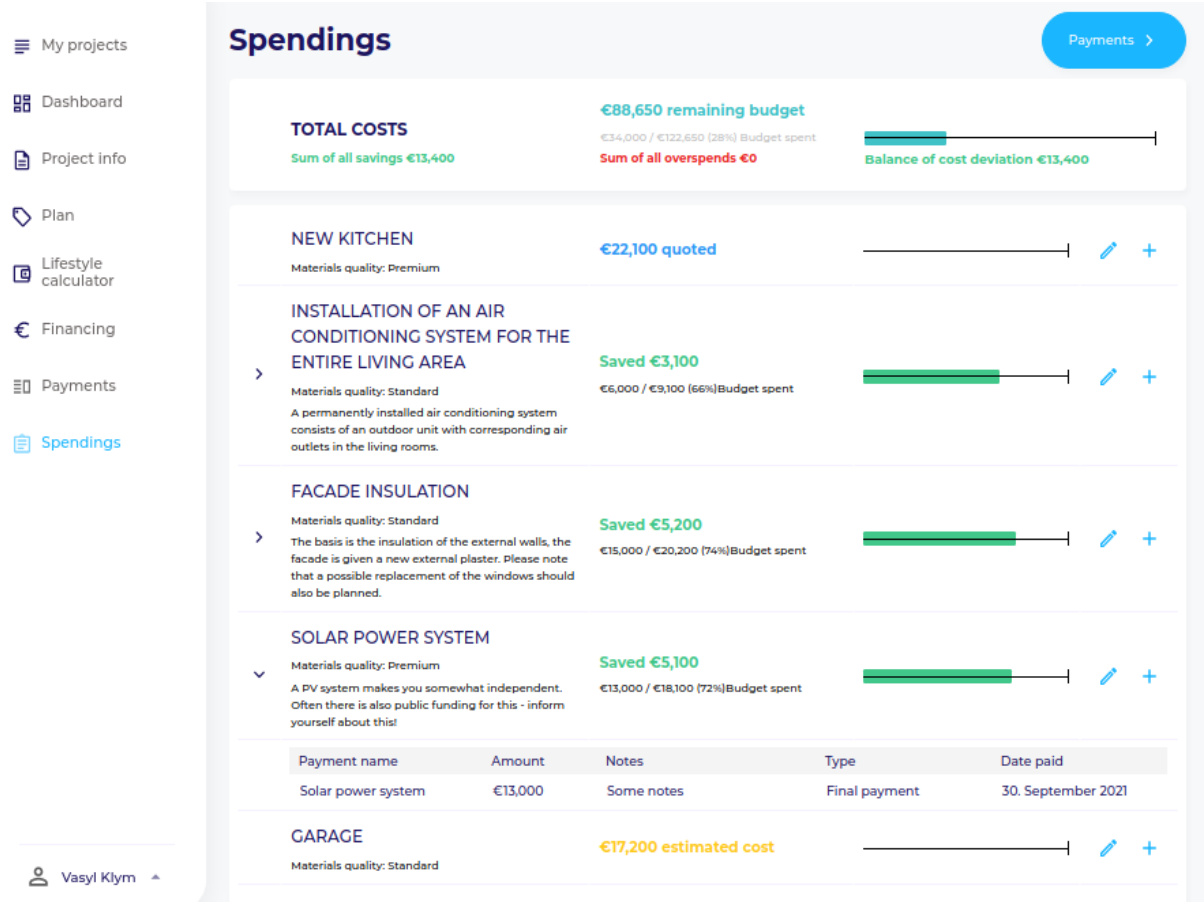



Рисунок 2.16 – Сторінка планування витрат

Користувач має можливість переглянути загальну суму всіх платежів та розраховану вартість проекту. Відношення запланованої вартості категорії витрат до здійснених витрат представлено графіком, у якому чорна шкала показує заплановану вартість, зеленим кольором показано вартість здійснених платежів. На даній сторінці користувач має можливість змінити заплановану вартість витрат по кожній з категорій, також має можливість додати платіж та переглянути всі платежі до кожної з категорій.

# 3 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

## 3.1 План тестування

План тестування описує підхід до тестування і загальну структуру, яка буде сприяти тестуванню. Мета тесту – перевірити, що функціональність модулів виконується у відповідності зі специфікаціями. Тест виконає і перевірить тестові сценарії, визначить, виправить і повторно протестує всі дефекти високого та середнього ступеня важливості у відповідності з критеріями входу, визначить пріоритети дефектів більш низького ступеня важливості для майбутнього виправлення.

Кінцевий результат тестування забезпечує: готове до випуску програмне забезпечення; набір стабільних тестових сценаріїв, які можуть бути повторно використані для виконання функціональних тестів.

План принципів тестування включає:

– зосередження тестування на досягненні бізнес-цілей, економічної ефективності і якості;

– узгодження спільних процедур для всіх команд, які підтримують діяльність з тестування;

– чітке визначення процесів тестування, але при цьому гнучкі, з можливістю зміни в мірі необхідності;

– заходи з тестування ґрунтуються на попередніх етапах, щоб уникнути дублювання зусиль;

– середовище тестування і дані максимально близькі до виробничого середовища та даних;

– тестування розділене на окремі етапи, кожен з яких має чітко визначені цілі завдання.

Для тестування системи визначено два цикли випробувань, кожен з яких буде виконувати всі сценарії. Перший цикл полягає у виявленні будь-яких блокувань,

критичних дефектів і більшості серйозних дефектів. Очікується, що він буде використовувати деякі обхідні шляхи, щоб дістатися до всіх сценаріїв. Мета другого циклу полягає в тому, щоб виявити залишилися високі і середні дефекти, усунути обхідні шляхи з першого циклу, виправити прогалини в сценаріях і отримати результати продуктивності.

Помилки, які виявляються в ході тестування веб-орієнтованої системи класифікуються у відповідності категоріям. Перелік рівнів важливості помилок наведено у таблиці 3.1.

Таблиця 3.1 – Категорії важливості помилок

| Рівень важливості помилки | Характерні особливості |
|---|---|
| Критичний | 1. Помилка даного рівня є досить критичною, щоб привести до потенційної втрати даних, збою системи або пошкодження файлів.<br>2. Помилка призводить до підвисання системи і вимагає перезавантаження системи. |
| Високий | 1. Помилка даного рівня призводить до некоректного або повної відсутності виконання основних функцій системи. |
| Середній | 1. Помилка погіршує якість системи. Проте існує обхідний шлях для досягнення бажаної функціональності.<br>2. Помилка перешкоджає тестуванню інших частин продукту. Проте інші частини можуть бути протестовані незалежно. |
| Низький | 1. Повідомлення помилки є недостатньо зрозуміле, однак має мінімальний негативний вплив на використання системи. |

Функціональне тестування системи буде містити попередньо завантажені тестові дані, які використовуються для тестування.

Виділимо такі рівні тестування: дослідний, функціональний, приймальний.

Дослідний рівень виконується на початку кожного циклу тестування. Його мета полягає в тому, щоб переконатися, що критичні дефекти усунені.

Даний рівень включає дослідницьке тестування проводиться в додатку без будь-яких тестових сценаріїв і документації.

Функціональний рівень включає здійснення функціонального тестування для перевірки функцій додатку. Функціональне тестування виконується шляхом подання вхідних даних і перевірки вихідних даних з системи. Тест буде виконуватися у відповідності з функціональними сценаріями. Тестування на даному рівні виконується після завершення дослідницького тесту. Затверджений документ функціональної специфікації, документи по варіанту використання доступний до початку етапу проектування тестування. Тестові випадки повинні бути затверджені та підписані до початку виконання тесту.

Тестування на приймальному рівні спрямоване на перевірку бізнес-логіки. Це дозволяє кінцевим користувачам завершити один остаточний огляд системи перед розгортанням. Оскільки користувачі найбільш схильні надавати інформацію про бізнес потреби і про те, як система адаптується до них, може статися так, що користувачі виконають певну перевірку, яка не міститься в сценаріях. На даному рівні команда виконує тестування на основі даних, отриманих від кінцевих користувачів та аналітиків. Тільки після завершення цього випробування продукт може бути випущений у виробництво.

## 3.2 Розробка тестів

Модульне або юніт-тестування є першим рівнем тестування та часто виконується самими розробниками.

Це процес підтвердження того, щоб окремі компоненти програмного додатку на рівні коду працюють відповідно до технічним завданням з його розробку.

Розробники в середовищі, керованому тестами, зазвичай пишуть та запускають тести перед тим, як ПЗ буде передано групі тестування.

Модульне тестування полегшує налагодження, тому що при виявлення проблем на ранній стадії для їх усунення потрібно менше часу, ніж при виявленні на пізніших стадіях тестування програми.

Практично модульне тестування полягає у написанні фрагмента коду для перевірки коду модуля, написаного для реалізації вимог.

На рисунку 3.1 зображено цикл модульного тестування.

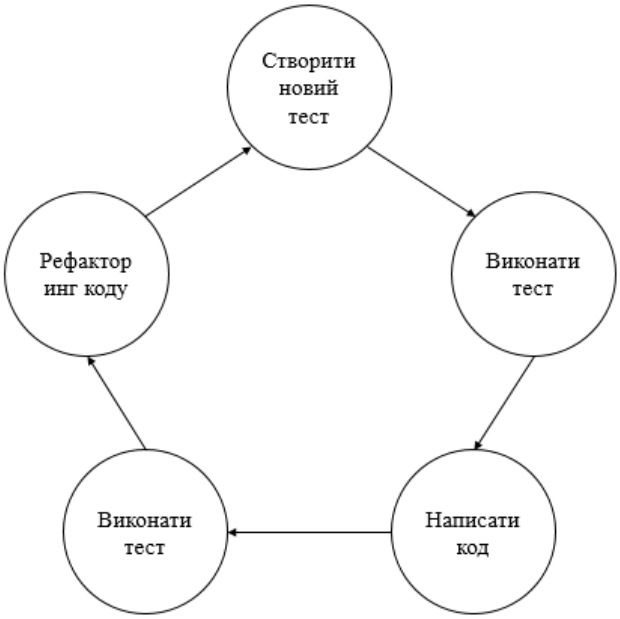



Рисунок 3.1 – Цикл модульного тестування ПЗ

Переваги модульного тестування:

- тестування можна проводити на ранніх етапах життєвого циклу розробки програмного забезпечення, коли інші модулі можуть бути недоступними для інтеграції;
- виправлення проблеми в модульному тестуванні може запобігти проблемам, що виникають на пізніших етапах розробки та тестування;
- вартість виправлення дефекту, виявленого в модульному тестуванні, значно нижчому від вартості, виявленої в системі або приймальних випробуваннях;
- повнота коду може бути продемонстрована за допомогою модульних тестів. Це корисніше у гнучкому процесі;
- підвищення надійності проектування та розробки, оскільки розробники пишуть тестові приклади, спочатку розбираючись у специфікаціях.

- дозволяє просто визначити винуватця помилки;
- економить час розробки: завершення коду може зайняти більше часу, але через зменшення кількості дефектів можна заощадити загальне час розробки.

Незважаючи на те, що модульне тестування може проводитись вручну, автоматизація цього процесу прискорить цикли доставки та розширить охоплення тестування.

Для тестування системи було вибрано фреймворк Jest, що є універсальною платформою для тестування з можливістю адаптації до будь-якої бібліотеки чи фреймворку JavaScript. На рисунку 3.2 наведено реалізацію модульного тесту для об’єднання проектів.

```
it('check if the function successfully merges user and guest projects', async () => {
    expect.assertions(2);
    const registeredUserId = uuid();
    const guestId = uuid();

    await this.populateProjects(2, registeredUserId);
    await this.populateProjects(2, guestId);

    const { formData } = this.budgetCalculatingTestData[0];
    await this.budgetService.budgetCalculator({ formData, userId });

    const loanData = { budget: 550000, price: project.price, period: 20, date: new Date(), interest_type: 'variable' };

    await this.projectService.mergeGuestProjects(guestId, registeredUserId);

    const numberOfUserProjects = await this.projectModel.countDocuments({ created_by: registeredUserId });
    const numberOfGuestProjects = await this.projectModel.countDocuments({ created_by: guestId });

    expect(numberOfUserProjects).toBe(4);
    expect(numberOfGuestProjects).toBe(0);
});
```

Рисунок 3.2 – Програмний код тестування методу об’єднання проектів гостя і зареєстрованого користувача

Тестова функція it приймає два аргументи: перший аргумент – рядок, який описує дію конкретного тесту; другий аргумент – функція зворотнього виклику, що містить функцію очікування та функцію збігу.

Тести пишуться всередині функції describe, яка групує тести. Створення групи тестів наведено у рисунку 3.3.

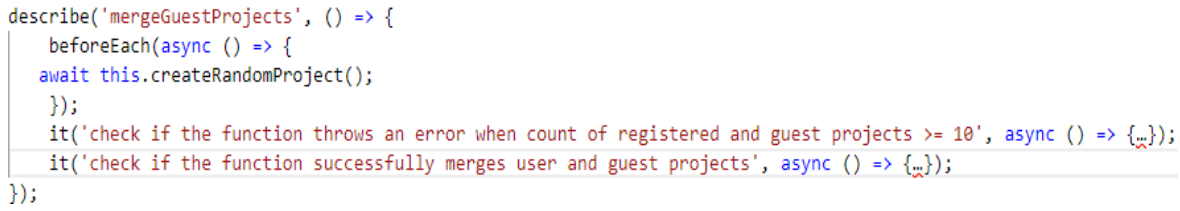



Рисунок 3.3 – Програмний код створення групи тестів

Функція describe приймає два аргументи: перший аргумент – рядок, який з'являється в терміналі під час виконання тестів який описує тестовий блок; другий аргумент – функція зворотнього виклику, яка містить окремі тести. Часто при написанні тестів нам потрібно зробити деяку роботу до того, як запуститься тест, і деяку роботу з його завершення. Jest надає допоміжні функції для цього. Функція describe містить окрім тестів також хук beforeEach, який виконується перед запускою кожного з тестів в даному блоці. Окрім хука beforeEach Jest містить такі хуки: beforeAll – виконує функцію перед виконанням будь-якого з тестів у цьому файлі; afterAll – запускає функцію після завершення всіх тестів у цьому файлі; afterEach – запускає функцію після завершення кожного з тестів у цьому файлі.

Інтеграційне тестування – це фаза тестування програмного забезпечення, під час якої окремі модулі програми комбінуються та тестуються разом, у взаємодії [13]. Даний вид тестування представляє собою рівень тестування програмного забезпечення, де окремі блоки/компоненти об'єднуються та тестуються як група. Метою цього рівня тестування є виявлення несправностей у взаємодії між інтегрованими блоками.

Цей тип тестування зосереджується на взаємодії та інтерфейсах між системами. Інша система, з якою необхідно інтегрувати, може бути внутрішньою або зовнішньою. Такі системи зазвичай відкриваються через інтерфейс прикладного програмування (API) або мікросервіс. Інтеграційні тести проводяться розробниками або незалежними тестувальниками та зазвичай складаються з комбінації автоматизованих функціональних та ручних тестів.

Основна функція або мета цього тестування – перевірити інтерфейси між модулями. Окремі модулі спочатку тестуються ізольовано. Відразу після того, як

модулі проходять модульне тестування, вони інтегруються один з одним, щоб перевірити їхню комбінаційну поведінку та правильність реалізації вимог. На рисунку 3.4 наведено інтеграційний тест, який на клієнті редагує дані плану проекту.

```
it('Edit project plan template', async () => {
    const plan = Factory.build(projectPlan, { });
    const data = { plan };
    withRenderedTemplate('editProjectPlanModal', data,
    el => {
    var $tmpl = $(el);
    chai.assert.equal($tmpl.find('#edit-project-plan-modal').length, 1);

    chai.assert.equal($tmpl.find('#project-plan-key').val(), plan.title);
    chai.assert.equal($tmpl.find('#rotation').val(), plan.description);
    chai.assert.equal($tmpl.find('#project-plan-type.val(), plan.hashTags);
    chai.assert.equal($tmpl.find('#projectplansuperstructure').val(), plan.imageIds);

    var newProjectPlanKey = 'Project plan test 2';
    $tmpl.find('#title').val(newProjectPlanKey);
    $tmpl.find('form').submit();
    setTimeout(() => {
    var updatedProjectPlan = await projectPlanModel.findOne(publication._id);
    chai.expect(updatedProjectPlan.key).to.equal(newProjectPlanKey);
}, 2000);
}); });
```

Рисунок 3.4 – Програмний код інтеграційного тесту для редагування плану проекту

На наведеному лістингу створюємо шаблон редагування плану проекту. Після чого виконуємо перевірку чи було коректно заповнено даними з моделі всі поля форми. Наступним кроком виконуємо псевдо зміну значення поля та виконуємо подання форми. Для імітування виклику асинхронного серверного методу використовуємо метод setTimeout. Після подання форми буде отримано документ з колекції та виконано перевірку чи поле було змінено.

Отже, було досліджено модульний та інтеграційний методи тестування системи та реалізовано детальні сценарії автоматичного тестування

# 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКИ ЖИТТЄДІЯЛЬНОСТІ

## 4.1 Охорона праці

Тема кваліфікаційної роботи магістра передбачає розробку веб-орієнтованої системи планування витрат у сфері виробництва на основі JavaScript технологій. Інженер-програміст в процесі розробки програмного забезпечення використовує персональні електронно-обчислювальні машини. Тому виконання правил з охорони праці є важливим фактором для запоруки безпеки та здоров‘я.

Працюючи за персональний комп’ютером ступінь розумової напруги значно зростає. Навантаження також спричиняє емоційне вигорання, високим показником напруженості зору і нагрузкою на м‘язи рук при використанні електро-обчислювальної машини – клавіатури. Отже, правильна організація робочого простору є важливим та доцільним елементом, для забезпечення якого необхідно уникати незручності в розташуванні засобів праці. В певній мірі, підвищення ефективності і результативності праці робітника залежить від факторів втомлюваності чи бадьорості. Згідно з Державними санітарними правилами і нормами роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98 відповідність розташування та організація робочого місця з ВДТ повинна відповідати ергономічними вимогами, враховуючи особливість трудової діяльності [14].

При розробці системи оцінки вартості витрат у сфері будівництва повинні враховуватись існуючі санітарні нормативи освітлення, вимоги до параметрів мікроклімату (температура, відносна вологість), ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електромагнітного, ультрафіолетового та інфрачервоного полів. Конкретні показники зазначених санітарних норм є у інструкції “Державні санітарні правила і норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин” ДСанПіН 3.3.2.007-98 [15].

Формулювання і аналіз загрозливих і шкідливих чинників виробництва необхідно основувати на аналізі з дотримання вимог, що установлені наказом “Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями” [16]. Для об‘єктивного оцінювання умов праці робочого місця здійснюється його атестація, а також «Гігієнічна класифікація праці». Дані показників загрозливих, шкідливих чинників залежать від самого виробничого середовища та гостроти напруження трудового процесу.

Згідно Гігієнічної класифікації праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу» умови праці поділяються на 4 класи [17]. Умови праці Під час розробки програмного забезпечення було дотримано оптимальних умови праці, які належать до 1 класу умов праці, за яких зберігається не лише здоров’я працівників, а й створюються передумови для підтримання високого рівня працездатності.

Розробка комерційного програмного продукту відбувається в офісному приміщенні, в якому знаходяться комп’ютеризовані робочі місця та експлуатується комп’ютерна техніка інженерами-програмістами. Під час планування приміщення слід дотримуватись державних нормативно-правових актів та законів. Згідно ДБН В.2.5-28:2018 приміщення, в якому розробляється система оцінки вартості житлового будівництва, повинно мати природне і штучне освітлення [18]. Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід, і забезпечувати коефіцієнт природної освітленості (КПО) не нижче, ніж 1,5%. Також у приміщеннях глибиною 6 м та більше доцільно застосовувати на вікнах спеціальні світловідбивні екрани та жалюзі, що перерозподіляють світловий потік в глибину приміщення. Оскільки будинок розташований у відносній віддаленості від прилеглих будівель, то які-небудь перешкоди природному освітленню розглянутого приміщення відсутні.

Під час роботи за комп’ютером інженер-програміст піддається впливу небезпечних та шкідливих здоров’ю чинників. Психофізіологічні чинники: напруження уваги, пам’яті та зору; значна кількість інформації, яку необхідно обробити за одиницю часу; в деяких випадках монотонність завдань. Фізичні

чинники: збільшений рівень інфрачервоного, ультрафіолетового та рентгенівського випромінювання; відсутність рівномірного розподілу яскравості на робочому місці.

Для зменшення впливу негатичних чинників, під час розробки програмної системи, було правильно розміщено робочі місця та комп'ютерне обладнання. Це питання регулюється широким спектром нормативно-правових актів. Законом України "Про охорону праці" передбачено обов'язки роботодавця, які описують необхідність забезпечення безпечних та комфортних умов праці та права працівників на отримання таких умов здійснення роботи [19].

Під час розробки, тестування та впровадження інформаційної системи були дотримані всі вимоги, норми та державні стандарти з охорони праці.

## 4.2 Безпека в надзвичайних ситуаціях

### 4.2.1 Підвищення стійкості роботи будівельних підприємств у воєнний час

Підвищення стійкості роботи підприємств будівельної галузі, у воєнний час – одна із основних задач цивільної оборони України. Могутність країни базується на стійкій економіці. В сучасних умовах, коли науково-технічний прогрес у всіх сферах виробництва досяг небачених масштабів і привів до створення зброї масового ураження, в разі розгортання великомасштабної війни основні промислові центри і райони будуть головною ціллю для знищення зі сторони противника. Адже виведення економіки з ладу може призвести до того, що країна не зможе стояти на оборонні своїх кордонів та підтримувати життєдіяльність населення. На сьогодні, через бойові дії на сході України (Війни на Донбасі), проблема підвищення стійкості роботи підприємств будівельної галузі стоїть як ніколи гостро.

Будівельне підприємство – це підприємство, яке діє в сфері будівництва і здійснює наукові, експериментальні, вишукувальні та проектні роботи, видобуток

сировинних ресурсів і їх переробку, виготовлення матеріалів, виробів і конструкцій, зведення всіх видів будівель і споруд, транспортне обслуговування. Діяльність будівельних підприємств забезпечується наявністю в їх розпорядженні необхідних ресурсів: людських, фінансових, матеріальних, енергетичних, за допомогою яких створюється продукція. Одним з основних показників виробничої діяльності будівельного підприємства є продукція будівельного підприємства – це матеріальні цінності, створені в результаті діяльності будівельного підприємства. Продукція може ставитися до категорії «кінцевої» (закінчені і здані в експлуатацію будівлі і споруди) або до «проміжної» – вироби підприємств будівельної індустрії, окремі види робіт, частини будівель і ін.

Будівництво як галузь економіки бере участь у створенні основних фондів (будівель та споруд) для всіх галузей національного господарства, тобто створює умови для виробничого процесу.

Під стійкістю роботи підприємств будівельної галузі розуміють їх здатність за умов дії надзвичайних ситуацій виробляти продукцію в запланованих обсязі та номенклатурі, а при одержанні слабких чи середніх руйнувань чи порушенні постачання сировини відновлювати своє виробництво в мінімально короткі терміни. Щоб забезпечити нормальну роботу під час війни промислових об’єктів будівництва, скоротити можливі матеріальні втрати, необхідно ще в мирний час виконати великий комплекс різних заходів, які забезпечили б їхнє функціонування. Ці заходи спрямовані на зниження можливих втрат і руйнувань від сучасних засобів ураження і створення умов для нормальної роботи підприємств як у воєнний, так і в мирний час.

На стійкість роботи об’єктів будівництва впливають такі фактори:

– надійність захисту робітників від дії вражаючих факторів, що виникають під час надзвичайних ситуацій;

– здатність будівельного комплексу протистояти дії вражаючих факторів;

– надійність систем постачання об’єкта сировинною для виробництва певного виду продукції;

– захищеність об’єкта від дії вторинних вражаючих факторів.

При вирішенні проблеми підвищення стійкості роботи підприємств будівельної галузі керуються єдиними принциповими положеннями:

– завчасне проведення заходів цивільного захисту, спрямованих на зниження можливих втрат та руйнувань у разі застосування збоку противника зброї масового ураження і на створення умов для швидкого відновлення виробництва після часткового руйнування;

– комплексний підхід в розробці і здійсненні заходів для всіх напрямків діяльності підприємства;

– узгодження цих заходів з територіальними і військовими органами управління.

Заходи з підвищення стійкості плануються з урахуванням місцевих умов, ступеня важливості об’єкта, його географічного положення, економічної доцільності проведення заходів. На мирний час планують, в основному, трудомісткі заходи, які потребують значних матеріальних витрат і часу, а на період загрози виникнення НС – такі заходи, які не потребують значних затрат часу чи проведення яких не є доцільним при нормальному функціонуванні. Також при проведені заходів з ЦЗ потрібно враховувати і внутрішні фактори, що впливають на стійкість: розмір виробництва, виду продукції, що випускається, чисельність працівників, рівень їх дисциплінованості і компетентності, особливості технології виробництва, системи постачання виробництва сировиною, технічною і питною водою, газо- та електроенергією.

З урахуванням розглянутих вище факторів виділяють такі основні шляхи і способи підвищення стійкості роботи підприємств будівельної галузі:

– забезпечення надійного захисту робітників і службовців: укриття робітників і службовців, які продовжують роботу на об'єкті у воєнний час;

проведення евакуації робітників, службовців і членів їх сімей та забезпечення їх життєдіяльності; використання індивідуальних засобів захисту;

– захист основних виробничих фондів об'єкта від поразки: підвищення певною мірою опірності будівель, споруд впливу ударної хвилі, світлового випромінювання; укриття найбільш уразливого обладнання в захисних пристроях (шатрах, камерах, конусах і ін.); часткову зміну технології виробництва; вивезення в безпечні райони надлишків горючих речовин;

– забезпечення сталого постачання об'єкта всім необхідним для виробництва: підвищення надійності роботи транспорту; підготовка паливноенергетичного господарства до роботи у воєнний час;

– підвищення надійності та оперативності управління виробництвом: створення об'єктового і заміського пункту управління; прокладка підземних кабельних ліній зв'язку до всіх елементів об'єкта; створення оперативних змін управління для основного і заміського пунктів управління;

– підготовка до виконання робіт по відновленню об'єкта у воєнний час: планування відновлювальних робіт за кількома варіантами; підготовка ремонтних бригад; створення необхідного запасу матеріалів і обладнання, надійний його захист; створення страхового фонду технічної документації.

Кожен шлях містить кілька способів підвищення стійкості роботи підприємства, які, в свою чергу, містять кілька заходів ЦЗ або доповнюються ними. Наведені вище шляхи підвищення стійкості підприємств будівельної галузі реалізуються за допомогою затверджених норм з ЦЗ прийнятих і обов’язкових до виконання для всіх об’єктів усіх галузей виробництва не залежно від форм власності і підпорядкування. Норми ЦЗ призначені для:

– зменшення рівня руйнувань основних фондів виробництва;

– підвищення стійкості роботи об’єкта і галузей виробництва;

– забезпечення умов для ліквідації наслідків надзвичайних ситуацій;

Контроль за виконанням вимог згаданих норм покладається на Управління та відділи з питань надзвичайних ситуацій.

# ВИСНОВКИ

У результаті виконання кваліфікаційної роботи магістра було здійснено аналіз предметної області, визначено мету, задачі, головні вимоги до проектування та розробки програмного забезпечення для планування витрат у сфері будівництва, сформовано структуру робіт проекту, сформовано календарний план виконання робіт.

Було виконано мету роботи, а саме здійснено аналіз предметної області, спроектовано архітектуру та реалізовано систему планування витрат у сфері будівництва на основі JavaScript технологій. Розроблений застосунок дозволяє якісно та швидко обчислити енергоефективність будинку, скласти план проекту стосовно проведення будівельних робіт для збільшення енергоефективності будинку та зменшення викидів $CO_2$, розрахувати особистий потенціал заощаджень. Система надає можливість точно описувати проект користувача і здійснювати оцінку вартості витрат по кожній із запланованих категорій витрат на будівельні роботи.

Визначено функціональні та нефункціональні вимоги до системи, здійснено структурно-функціональне моделювання, спроектовано діаграму варіантів використання та діаграму діяльності.

Здійснено аналіз та огляд існуючих стеків веб-технологій для розробки програмного забезпечення. Системи розроблена за допомогою бази даних MongoDB, що пришвидшило роботу додатку, оскільки дана база даних є найкращим вибором для виконання складних запитів для документів із складною структурою; фреймворку для веб-додатків Express; бібліотеки JavaScript для створення користувацьких інтерфейсів React; кросплатформного середовища виконання JavaScript коду Node.js; для стилів використано препроцесор SCSS. В якості середовища розробки було вибрано оптимізований редактор коду для сучасних веб-додатків Visual Studio Code.

Спроектовано архітектуру системи на основі трирівневої клієнт-серверної архітектури, розроблено серверну частину веб-застосунку, яка здійснює складні обчислення та витримує значні навантаження, відповідає за обробку запитів і реалізацію функціоналу. Написано інтеграційні та модульні тести за допомогою фреймворку Jest.

Результатом кваліфікаційної роботи магістра є готова веб-орієнтована система для планування витрат у сфері будівництва. В перспективі передбачається вдосконалення системи, додавання нового функціоналу: додавання нових типів проектів, інтеграція Google Analytics для збору статистики про відвідуваність сайту, та активність користувачів на ньому, імплементація функціоналу для адміністратора, який матиме можливість переглядати створенні проекти, генерувати звіти у форматі Excel та відповідати на електронні листи від користувачів розробленої системи.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Енергоефективний будинок крок за кроком – Київ: Інститут місцевого розвитку, 2012. – 144 с.
2. Пасивний будинок [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Пасивний_будинок
3. Тимофєєв М. Енергоефективний панельний житловий будинок. Архітектура будівель та споруд / М. Тимофєєв, Г. Гетун., 2018. – 190 с.
4. Концепція національної екологічної політики України на період до 2020 року [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/880-2007-%D1%80#Text
5. Енергетична Революція — із Стандартом Пасивного Будинку [Електронний ресурс] – Режим доступу до ресурсу: https://passivehouse-igua.com/passive-house/energy-revolution-with-passive-house-standard/
6. Управління розробкою програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Управління_розробкою_програмного_забезпечення
7. Діаграма діяльності [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_діяльності
8. Мова програмування JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript
9. Платформа Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Node.js
10. База даних [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/База_даних
11. Google Storage [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Google_Storage
12. Хмарна PaaS-платформа Heroku [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Heroku

13. Інтеграційне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/інтеграційне_тестування

14. Жизненный цикл проекта [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Жизненный_цикл_проекта

15. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПІН 3.3.2.007-98 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text

16. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508-18#Text

17. Державні санітарні правила та норми «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/ z0472-14#Text

18. Державні будівельні норми України / Природнє і штучне освітлення [Електронний ресурс] – Режим доступу до ресурсу: https://dbn.co.ua/load/normativy/dbn/dbn_v_2_5_28/1-1-0-1188

19. Закон України "Про охорону праці" [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12#Text

20. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – Інженерія програмного забезпечення (Освітньопрофесійна програма – «Програмне забезпечення систем», Освітньонаукова програма – «Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020-51с.

# ДОДАТКИ

Додаток А – Технічне завдання

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
Кафедра “Програмної інженерії”

# ТЕХНІЧНЕ ЗАВДАННЯ

на виконання кваліфікаційної роботи магістра
«Розробка веб-орієнтованої системи планування витрат у сфері будівництва на основі JavaScript технологій»

Керівник роботи:
асистент Мудрик І.Я.
“___” _____ 2021р.
_____

Виконавець:
студент групи СПм-61
Клим Василь Васильович
“___” _____ 2021р.
_____

Тернопіль 2021

ЗМІСТ

# 1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки магістрів за спеціальністю «Інженерія програмного забезпечення» 2021-2022 н.р.

Тема кваліфікаційної роботи магістра: «Розробка веб-орієнтованої системи планування витрат у сфері будівництва на основі JavaScript технологій».

Термін виконання: до __.___._____р.

# 2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Для розробки веб-орієнтованої системи планування витрат у сфері будівництва на основі JavaScript технологій, було використано мову програмування JavaScript бібліотеку React, серверний фреймворк Express.js, об’єктно-орієнтовану мову програмування JavaScript. В якості СКБД для даної системи було обрано MongoDB.

Актуальність енергозбереження та підвищення енергетичної ефективності будівель обумовлено високими витратами та постійним зростанням тарифів на енергоресурси.  Цілеспрямована реалізація програм енерго-ресурсозбереження дозволила б при значно менших, ніж для введення нових енергетичних потужностей, капітальних витрат, знизити дефіцит енергії та створити сприятливі умови для вирішення проблеми у паливно-енергетичному комплексі.

Веб-орієнтована система планування витрат у сфері будівництва призначена для обчислення енергоефективності будинку і планування заходів щодо її підвищення та покращення будинку; можливості створення та ведення плану проекту із оцінкою вартості кожного з покращень. Після етапу плануваннння проекту система надає можливість вести облік здійснених платежів по кожній з категорій.

# 3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Функціональні вимоги

Система повинна надавати користувачу такі можливості:

- здійснення гостьового входу;
- реєстрація в системі за допомогою сервісів: Google, Facebook;
- виконання оцінки енергоефективності будинку;
- обчислення кількості викидів CO2 до планування та потенційно можливої кількості після проведення ремонтних робіт;
- обчислення кількості спожитої електроенергії до та після виконання робіт щодо збільшення енергоефективності будинку;
- створення проекту та плану;
- розрахунок фактичних витрат на ремонт житла, можливого потенціалу фінансової економії;
- редагування проекту;
- можливість отримання особистого консультування з надсиланням запиту для отримання кредиту;
- зміна мови інтерфейсу;
- експортування проектних розрахунків у форматі PDF;
- можливість заповнення форм із параметрами, які описують проект;
- коригування розрахованого плану;
- ведення обліку витрат по кожній із запланованих категорій;

Вхідна інформація отримується шляхом введення інформації користувачами.

Вихідна інформація:

- інформацію про план проекту відображається у вигляді графіків та діаграм;
- подається у файлі формату PDF та з можливістю його завантаження.

## 3.2 Технічні вимоги

Вимоги до серверної частини:  ОС Linux, Windows не менше ніж 4Гб ОЗП;

Вимоги до клієнтської частини: Наявність браузера, пристрої вводу і виводу інформації;

Додаткові вимоги: наявне підключення до мережі Інтернет, автоматичне резервування, забезпечення одночасної роботи до 1000 клієнтів.

## 3.3 Програмні вимоги

Використання СУБД: MongoDB.

Розробка клієнтської частини: мова програмування JavaScript, бібліотека React.

Розробка серверної частини: платформа Node.js, фреймворк Express.js.

Середовище розробки: Microsoft VSCode.

# 4 СТАДІЇ НАПИСАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Написання кваліфікаційної роботи проводиться в наступному порядку:

- вибір та затвердження теми
- аналіз предметної області
- постановка задач розробки
- проектування діаграми варіантів використання
- побудова діаграми діяльності
- вибір СКБД та опис її фізичної моделі
- опис програмної реалізації
- опис схем використання програмного забезпечення
- тестування програмного забезпечення
- написання розділу «Охорона праці»
- написання розділу «Безпека в надзвичайних ситуаціях»
- оформлення записки кваліфікаційної роботи магістра
- попередній захист
- нормоконтроль
- захист кваліфікаційної роботи магістра

Результати виконання кожного етапу кваліфікаційної роботи магістра погоджуються з керівником роботи.

# 5 СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для кваліфікаційної роботи магістра повинні бути розроблені наступні документи:

- записка кваліфікаційної роботи;
- презентація;
- рецензія на кваліфікаційну роботу магістра;
- відгук керівника на кваліфікаційну роботу магістра;
- авторська довідка;
- протокол аналізу звіту подібності керівником роботи;
- диск з кваліфікаційною роботою магістра.

Записка кваліфікаційної роботи магістра оформляється згідно діючих вимог до нормоконтролю.

# 6 ПОРЯДОК ЗДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ МАГІСТРА

Розроблена системи повинна відповідати вимогами, що складаються з перерахованих у 3 розділі цього документу характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у розділі 5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в розділі 1 цього документу.

# 7 ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ В РОБОТІ

| Назва етапу | Відмітка* |
|---|---|
| Вибір та затвердження теми | |
| Аналіз предметної області | |
| Постановка задач розробки | |
| Проектування діаграми варіантів використання | |
| Побудова діаграми діяльності | |
| Вибір СКБД та опис її фізичної моделі | |
| Опис програмної реалізації | |
| Опис схем використання програмного забезпечення | |
| Тестування програмного забезпечення | |
| Написання розділу «Охорона праці» | |
| Написання розділу «Безпека в надзвичайних ситуаціях» | |
| Оформлення записки кваліфікаційної роботи магістра | |
| Попередній захист | |
| Нормоконтроль | |
| Захист кваліфікаційної роботи магістра | |

* відмітки про виконання етапу ставляться керівником проекту

Додаток Б – Публікація у науковому виданні

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ІВАНА ПУЛЮЯ**

**М А Т Е Р І А Л И**

**IX НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ**

# «ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

**8–9 грудня 2021 року**

**ТЕРНОПІЛЬ**
**2021**

УДК 004.45
**В.В. Клим - студент, І.Я. Мудрик – PhD**
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

# РОЗРОБКА ВЕБ-ОРІЄНТОВАНОЇ СИСТЕМИ ПЛАНУВАННЯ ВИТРАТ У СФЕРІ БУДІВНИЦТВА НА ОСНОВІ JAVASCRIPT ТЕХНОЛОГІЙ

UDC 004.45
**V.V. Klym student, I.I. Mudryk – PhD**

# DEVELOPMENT OF A WEB-BASED CONSTRUCTION COST PLANNING SYSTEM BASED ON JAVASCRIPT TECHNOLOGIES

Питання економії енергії є одним з найбільш поширених та актуальних питань цілого світу. В сфері будівництва вирішенням даного питання є проектування та будівництво пасивного будинку. В основі концепції пасивного будинку лежить поняття енергоефективності. Пасивний будинок – це сучасний будівельний стандарт, який відкриває абсолютно нові перспективи для архітекторів та інженерів. Промисловість позитивно реагує на потреби ринку пасивного будинку, розробляючи високоефективні, новітні продукти. Даний вид будинку пропонує реалістичне, рентабельне рішення для економічної будівлі, яке забезпечує високий рівень комфорту при використанні дуже малої кількості енергії для опалення та охолодження. В час глобального потепління та швидкого зростання цін на енергоносії – це будівля з майже нульовим енергоспоживанням. Даний стандарт будівництва заснований на десятиліттях науково-обґрунтованих даних і задоволених жителів. Стандарт пасивного будинку може бути запроваджений у всьому світі, залежно від місцевого клімату властивості окремих компонентів будуть відрізнятися, проте загальний підхід залишиться однаковий.

Здійснивши аналіз даної проблеми, виникає необхідність проектування та розробки програмного забезпечення для обчислення енергоефективності будинку, а також процесу планування будівельних робіт щодо підвищення її рівня. Користувачу пропонується заповнити дані форми, які необхідні для здійснення обчислень. На основі введених даних система виконує обчислення поточних та потенційних показників енергоефективності будинку. Після цього користувач зможе обрати бажані категорії витрат. Наступний крок виконання алгоритму: система запропонує пройти калькулятор способу використання житлових умов, який і обчислить потенційну кредитоспроможність. У випадку якщо розраховані витрати перевищуватимуть бюджет користувача, йому буде надано можливість отримати особисте консультування та надіслати запит для отримання кредиту. Програмне забезпечення надає можливість користувачу, під час здійснення витрат на будівельні роботи, вести облік по кожній категорії запланованих витрат. Отже, користувач отримує план створеного проекту, який включатиме всю інформацію про заплановані та фактичні витрати та буде доступний для експортування у форматі PDF.

Для розробки веб-орієнтованої системи використовується мова програмування JavaScript, бібліотека React, фреймворк Express. MongoDB використовується в якості системи керування базами даних.

**Література.**

1. Corner D. Passive House Details Solutions for High-Performance / D. Corner, J. Fillinger, A. Kwok., 2017. – 316 p.

## Лістинг коду 1 – Програмний код сторінки проектів користувача

```
import React from 'react';
import useProjects from 'hooks/project/useProjects';
import ProjectsList from 'components/ProjectList/ProjectsList';
import LoadingOverlay from 'components/common/LoadingOverlay';
import { Link } from 'react-router-dom';
import { Plus } from 'assets/icons';
import { useIntl } from 'react-intl';
import routePaths from 'router/route-paths';
import Button from 'components/common/Button';
import PageTitle from 'components/common/PageTitle';
import { Grid, Row, Col } from 'react-flexbox-grid';
import { Hidden, Visible } from 'components/common/Grid';
import Tooltip from 'components/common/Tooltip';
import CircleButton from 'components/common/CircleButton';
import WarningMessage from 'components/common/WarningMessage';

const AddProjectButton = ({ isMaxCountOfProjects, t }) => (
  <Link to={routePaths.newProject} style={{ pointerEvents: isMaxCountOfProjects ?
'none' : 'initial' }}>
    <Hidden xs sm>
      <Button disabled={isMaxCountOfProjects} startIcon={<Plus />}>
        {t({ id: 'my_projects.new_project' })}
      </Button>
    </Hidden>
    <Visible xs sm>
      <CircleButton color="primary" disabled={isMaxCountOfProjects}>
        <Plus />
      </CircleButton>
    </Visible>
  </Link>
);

export default function MyProjects() {
  const [projects, loading] = useProjects();
  const { formatMessage: t } = useIntl();
  const isMaxCountOfProjects = projects.length >= 10;

  if (loading) {
    return <LoadingOverlay />;
  }

  if (projects.length === 0) {
    return (
      <WarningMessage
        title={t({ id: 'page_titles.my_projects' })}
        message={t({ id: 'my_projects.create_project_message' })}
        btnLabel={t({ id: 'my_projects.new_project' })}
        btnLink={routePaths.newProject}
      />
    );
  }

  return (
    <Grid>
      <Row between="xs" middle="xs">
        <Col xs={8}>
```

```
          <PageTitle className="text-left">{t({ id: 'page_titles.my_projects'
})}</PageTitle>
        </Col>
        <Col className="d-flex justify-content-end" xs={4}>
          <Tooltip
            hide={!isMaxCountOfProjects}
            overlay={<p>{t({ id:
'my_projects.delete_project.tooltip_max_count_of_projects' })}</p>}
            placement="top"
          >
            <div>
              <AddProjectButton {...{ isMaxCountOfProjects, t }} />
            </div>
          </Tooltip>
        </Col>
      </Row>
      {!loading ? <ProjectsList {...{ projects }} /> : null}
    </Grid>
  );
}
```

Лістинг коду 2 – Програмний код для відправки даних форми створення проекту

```
import { useCallback } from 'react';
import { useApolloClient, useMutation } from '@apollo/client';
import gql from 'graphql-tag';
import { formatGraphqlErrors } from 'utils/helpers';
import { PROJECT_FRAGMENT } from 'graphql/fragments/project';
import { PROJECT_QUERY } from './useProject';
import { PROJECT_PLAN_QUERY } from '../useProjectPlan';
import { GET_CATEGORIES_QUERY } from '../payments/useGetCategories';

const UPDATE_PROJECT_NEW_BUILDING_MUTATION = gql`
  mutation updateProjectNewBuilding($projectId: ID!, $formData:
UpdateProjectNewBuildingInput!) {
    updateProjectNewBuilding(projectId: $projectId, formData: $formData) {
      ...project
    }
  }
  ${PROJECT_FRAGMENT}
`;

const useUpdateProjectNewBuildingMutation = () => {
  const [updateProjectNewBuildingMutation] =
useMutation(UPDATE_PROJECT_NEW_BUILDING_MUTATION);
  const client = useApolloClient();

  const mutate = useCallback(
    async (projectId, formData) => {
      try {
        const {
          data: { updateProjectNewBuilding: updatedProject },
        } = await updateProjectNewBuildingMutation({
          variables: { projectId, formData: { ...formData } },
          refetchQueries: [
            { query: PROJECT_PLAN_QUERY, variables: { projectId } },
            { query: PROJECT_QUERY, variables: { id: projectId } },
            { query: GET_CATEGORIES_QUERY, variables: { projects: [projectId] } },
          ],
          awaitRefetchQueries: true,
```

```
        });

        client.writeQuery({
          query: PROJECT_QUERY,
          variables: { id: projectId },
          data: { project: { ...updatedProject } },
        });
      } catch (error) {
        const e = formatGraphqlErrors(error);
        throw e;
      }
    },
    [updateProjectNewBuildingMutation, client],
  );
  return mutate;
};

export default useUpdateProjectNewBuildingMutation;
```

Лістинг коду 3 – Програмний код для відображення форми сторення проекту

```
import React, { useCallback } from 'react';
import { Grid } from 'react-flexbox-grid';
import StepWizard from 'react-step-wizard';
import useFormValuesRef from 'hooks/useFormValuesRef';
import RenovationHouseThirdStep from './tabs/RenovationHouseThirdStep';
import RenovationHouseFourthStep from './tabs/RenovationHouseFourthStep';
import RenovationHouseFifthStep from './tabs/RenovationHouseFifthStep';
import RenovationHouseSixthStep from './tabs/RenovationHouseSixthStep';
import RenovationHouseSeventhStep from './tabs/RenovationHouseSeventhStep';
import CO2CalculatorFormFirstStep from './tabs/co2-calculator-
form/CO2CalculatorFormFirstStep';
import Nav from '../Nav';
import s from '../calculators.module.scss';
import CO2StatusSecondStep from './tabs/CO2StatusSecondStep';
import CostOverview from '../CostOverviewTab';

const formValueMock = {
  name: 'My house',
  country: 'AT',
  zip: '1414',
  heated_living_area: 220,
  kind_of_house: 'single_family_house',
  energy_standard: 'between_1960_and_1975',
  renovations: ['new_windows', 'insulation_top_ceiling',
'insulation_basement_ceiling', 'insulation_facade'],
  is_solar_heating_system: true,
  solar_heating_system_type: 'heating_and_hot_water',
  number_of_people: 4,
  heating_system: 'natural_gas',
  age_of_heating_system: 'between_10_and_20_years',
  temperature: 'less_than_21',
  hot_water_producer: 'electric_boiler',
  amount_of_hot_water: 'medium',
  is_devices_younger_than_10_years: true,
  power_consumers: ['sauna', 'outdoor_whirlpool'],
  is_solar_power_system: true,
  solar_power_system_size: 10,
  solar_power_system_consuming_percentage: 15,
};

const tabs = [
```

```
  { component: CO2CalculatorFormFirstStep, label:
'project_details.tabs.house_info' },
  { component: CO2StatusSecondStep, label: 'project_details.tabs.co2_status' },
  { component: RenovationHouseThirdStep, label: 'project_details.tabs.common' },
  { component: RenovationHouseFourthStep, label: 'project_details.tabs.energy' },
  { component: RenovationHouseFifthStep, label: 'project_details.tabs.indoor' },
  { component: RenovationHouseSixthStep, label: 'project_details.tabs.outdoor' },
  { component: RenovationHouseSeventhStep, label: 'project_details.tabs.other' },
  { component: CostOverview, label: 'project_details.tabs.cost_overview' },
];

const RenovationHouseForm = ({ initialValues, onSubmit, isEditMode, step }) => {
  const formValuesRef = useFormValuesRef([], { formValueMock, initialValues });

  const onSubmitStep = useCallback(
    (values) => {
      formValuesRef.current = { ...formValuesRef.current, ...values };
    },
    [formValuesRef],
  );

  const onBack = useCallback(
    (values, callback) => {
      onSubmitStep(values);
      callback();
    },
    [onSubmitStep],
  );

  return (
    <Grid className={s.gridContainer}>
      <StepWizard initialStep={step} transitions={{}} isLazyMount nav={<Nav
isEditMode={isEditMode} tabs={tabs} />}>
        {tabs.map(({ label, component: C }) => (
          <C key={label} {...{ onSubmitStep, onBack, formValuesRef, initialValues,
onSubmit, isEditMode }} />
        ))}
      </StepWizard>
    </Grid>
  );
};

export default RenovationHouseForm;
```

Лістинг коду 4 – Програмний код першого кроку калькулятора енергоефективності будинку

```
import React from 'react';
import { Field } from 'formik';
import SliderWithTooltip from 'components/inputs/SliderWithTooltip';
import ButtonSelect from 'components/inputs/ButtonSelect';
import Timeline from 'components/inputs/Timeline';

const FirstSection = ({ values, initialValues, t, isEditMode, goToThirdStep }) =>
(
  <>
    <Field
      name="heated_living_area"
      component={SliderWithTooltip}
      label={t({ id: 'renovation_house_wizard.heated_living_area' })}
```

```
      min={25}
      units="m²"
      max={initialValues?.house_area || 300}
      description={
        isEditMode
          ? t(
              { id: 'renovation_house_wizard.heated_living_area_disclaimer' },
              {
                link_to_the_third_step: (
                  <button type="button" onClick={goToThirdStep} className="link">
                    {t({ id: 'renovation_house_wizard.link_to_the_third_step' })}
                  </button>
                ),
              },
            )
          : null
      }
    />

    <Field
      name="kind_of_house"
      component={ButtonSelect}
      contentInColumn
      inOneRow={false}
      label={t({ id: 'renovation_house_wizard.kind_of_house.name' })}
      options={[
        {
          icon: ({ altText }) => <SingleStandardHouse className="fill"
title={altText} />,
          label: t({ id:
'renovation_house_wizard.kind_of_house.single_family_house' }),
          value: 'single_family_house',
        },
        {
          icon: ({ altText }) => <Bungalow className="fill" title={altText} />,
          label: t({ id:
'renovation_house_wizard.kind_of_house.bungalow_or_complex_floor_plan' }),
          value: 'bungalow_or_complex_floor_plan',
        },
        {
          icon: ({ altText }) => <TownHouse className="fill" title={altText} />,
          label: t({ id: 'renovation_house_wizard.kind_of_house.town_house' }),
          value: 'town_house',
        },
        {
          icon: ({ altText }) => <SemiDetachedHouse className="fill"
title={altText} />,
          label: t({ id:
'renovation_house_wizard.kind_of_house.semi_detached_house' }),
          value: 'semi_detached_house',
        },
      ]}
    />
    <Field
      name="energy_standard"
      component={Timeline}
      label={t({ id: 'renovation_house_wizard.energy_standard.name' })}
      description={t({ id: 'renovation_house_wizard.energy_standard.description'
})}
      options={[
        {
          label: t({ id: 'renovation_house_wizard.energy_standard.before_1960' }),
          value: 'before_1960',
```

```
        },
        {
          label: t({ id:
'renovation_house_wizard.energy_standard.between_1960_and_1975' }),
          value: 'between_1960_and_1975',
        },
        {
          label: t({ id:
'renovation_house_wizard.energy_standard.between_1976_and_1990' }),
          value: 'between_1976_and_1990',
        },
        {
          label: t({ id: 'renovation_house_wizard.energy_standard.after_1990' }),
          value: 'after_1990',
        },
        {
          label: t({ id:
'renovation_house_wizard.energy_standard.low_energy_house' }),
          value: 'low_energy_house',
        },
        {
          label: t({ id: 'renovation_house_wizard.energy_standard.passive_house'
}),
          value: 'passive_house',
        },
      ]}
    />

    {['low_energy_house', 'passive_house'].includes(values.energy_standard) ? (
      <Field
        label={t({ id: 'renovation_house_wizard.renovations.name' })}
        name="renovations"
        contentInColumn
        multi
        inOneRow={false}
        component={ButtonSelect}
        options={[
          {
            icon: ({ altText }) => <NewWindows className="fill" title={altText}
/>,
            label: t({ id: 'renovation_house_wizard.renovations.new_windows' }),
            value: 'new_windows',
          },
          {
            icon: ({ altText }) => <InsulationTop className="fill" title={altText}
/>,
            label: t({ id:
'renovation_house_wizard.renovations.insulation_top_ceiling' }),
            value: 'insulation_top_ceiling',
          },
          {
            icon: ({ altText }) => <InsulationBasement className="fill"
title={altText} />,
            label: t({ id:
'renovation_house_wizard.renovations.insulation_basement_ceiling' }),
            value: 'insulation_basement_ceiling',
          },
          {
            icon: ({ altText }) => <InsulationFacade className="fill"
title={altText} />,
            label: t({ id: 'renovation_house_wizard.renovations.insulation_facade'
}),
            value: 'insulation_facade',
```

```
          },
          {
            icon: ({ altText }) => <Ventilation className="fill" title={altText}
/>,
            label: t({ id:
'renovation_house_wizard.renovations.controlled_living_space_ventilation' }),
            value: 'controlled_living_space_ventilation',
          },
        ]}
      />
    ) : null}
    <Field
      name="number_of_people"
      component={ButtonSelect}
      contentInColumn
      inOneRow={false}
      options={[
        {
          icon: ({ altText }) => <TwoPeople className="fill" title={altText} />,
          altText: t({ id: 'alt_text.renovation_house.number_of_people.two_people'
}),
          value: 2,
        },
        {
          icon: ({ altText }) => <ThreePeople className="fill" title={altText} />,
          altText: t({ id:
'alt_text.renovation_house.number_of_people.three_people' }),
          value: 3,
        },
        {
          icon: ({ altText }) => <FourPeople className="fill" title={altText} />,
          altText: t({ id:
'alt_text.renovation_house.number_of_people.four_people' }),
          value: 4,
        },
        {
          icon: ({ altText }) => <FivePeople className="fill" title={altText} />,
          altText: t({ id:
'alt_text.renovation_house.number_of_people.five_people' }),
          value: 5,
        },
      ]}
      label={t({ id: 'renovation_house_wizard.number_of_people' })}
    />
  </>
);
```

Лістинг коду 5 – Програмний код реалізації методу для отримання даних проекту із сервера

```
import { useCallback } from 'react';
import { useApolloClient, useMutation } from '@apollo/client';
import gql from 'graphql-tag';
import { PROJECT_FRAGMENT } from 'graphql/fragments/project';
import { PROJECT_PLAN_QUERY } from '../useProjectPlan';
import { PROJECT_QUERY } from './useProject';

const UPDATE_PROJECT_RENOVATION_HOUSE_MUTATION = gql`
  mutation updateProjectRenovationHouse($projectId: ID!, $formData:
UpdateProjectRenovationHouseInput!) {
    updateProjectRenovationHouse(projectId: $projectId, formData: $formData) {
```

```
      ...project
    }
  }
  ${PROJECT_FRAGMENT}
`;

export const PROJECT_QUERY = gql`
  query project($id: ID!) {
    project(id: $id) {
      ...project
    }
  }
  ${PROJECT_FRAGMENT}
`;

const useProject = (projectId) => {
  const { loading, data } = useQuery(PROJECT_QUERY, { variables: { id: projectId }
});

  return [data?.project, loading];
};


const useUpdateProjectRenovationHouseMutation = () => {
  const [updateProjectRenovationHouseMutation] =
useMutation(UPDATE_PROJECT_RENOVATION_HOUSE_MUTATION);
  const client = useApolloClient();

  const mutate = useCallback(
    async (projectId, formData) => {
      try {
        const {
          data: { updateProjectRenovationHouse: updatedProject },
        } = await updateProjectRenovationHouseMutation({
          variables: { projectId, formData: { ...formData } },
          refetchQueries: [{ query: PROJECT_PLAN_QUERY, variables: { projectId }
}],
          awaitRefetchQueries: true,
        });
        client.writeQuery({
          query: PROJECT_QUERY,
          variables: { id: projectId },
          data: { project: { ...updatedProject } },
        });
      } catch (error) {
        const e = error.graphQLErrors ? new Error(error.graphQLErrors.map((err) =>
err.message)) : error;
        e.raw = error || '';
        throw e;
      }
    },
    [updateProjectRenovationHouseMutation, client],
  );
  return mutate;
};

export default useUpdateProjectRenovationHouseMutation;
```

Додаток Г – Диск із кваліфікаційною роботою магістра