Міністерство освіти і науки України

Ніжинський державний університет імені Миколи Гоголя

Навчально-науковий інститут природничо-математичних, медико-біологічних наук та інформаційних технологій

Кафедра інформаційних технологій, фізико-математичних та

економічних наук

Освітня програма: Комп’ютерні науки

Спеціальність:122 Компʼютерні науки

**КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття освітнього ступеня *магістр*

«Візуалізація даних для інформаційної системи обліку поселення в гуртожиток»

студента Стовбухи Ростислава Станіславовича

**Науковий керівник:**

Фетісов Валерій Сергійович,

кандидат економічних наук, доцент

**Рецензент:**

Доктор технічних наук, професор І. В. Казачков.

доцент, кандидат технічних наук Т. А. Кресан

**Допущено до захисту:** \_\_\_\_ \_\_\_\_ 2024 р.

Завідувач кафедри

проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Казачков І.В.

Ніжин − 2024

ЗМІСТ

ВСТУП 3

РОЗДІЛ 1 5

1.1. Сучасні підходи до створення програмного забезпечення для автоматизації житлових процесів 5

1.2. Існуючі системи управління поселенням у гуртожитках 6

1.3. Недоліки існуючих рішень та обґрунтування необхідності створення GurtKEY 7

РОЗДІЛ 2 9

2.1 Загальні принципи розробки програмного забезпечення 9

2.2 Метод візуального програмування 9

2.3 Використані інструменти розробки 10

2.4 Архітектура системи 15

2.5 Етапи створення GurtKEY 15

РОЗДІЛ 3 27

3.1 Верстка інтерфейсу з Webflow 27

3.2 Взаємодія з базою даних 34

3.3 Реалізація основних функцій 36

3.4 Тестування і результати 66

3.5 Використання в реальних умовах 68

ВИСНОВКИ 70

ВСТУП

**Актуальність теми.** Управління студентськими гуртожитками є важливим елементом життєдіяльності університетів, забезпечуючи комфортні умови проживання студентів та ефективне адміністрування ресурсів. Традиційні підходи, які базуються на паперових записах або застарілих електронних таблицях, значно обмежують можливості оперативного обліку даних, моніторингу стану поселення, ведення статистики та виконання фінансового обліку.

Із розвитком цифрових технологій з'являється необхідність у сучасних рішеннях, які забезпечують автоматизацію цих процесів, підвищуючи їхню точність, надійність і доступність. Візуалізація даних у таких системах сприяє кращому розумінню інформації, швидшому прийняттю рішень та зменшенню кількості рутинних завдань для адміністрації гуртожитків.

**Об’єкт дослідження** – інформаційні системи для обліку та управління процесами поселення в гуртожитках.

**Предмет дослідження** – автоматизація процесів поселення на основі вебзастосунків із візуалізацією даних.

**Метою кваліфікаційної роботи** є розробка вебзастосунку для обліку поселення у студентському гуртожитку, що дозволяє автоматизувати ключові процеси, забезпечуючи візуалізацію стану кімнат та інтерактивну взаємодію користувача із системою.

Для досягнення поставленої мети визначено такі завдання:

* аналіз сучасних підходів до автоматизації обліку поселення в гуртожитках;
* проектування архітектури та інтерфейсу інформаційної системи;
* реалізація вебзастосунку із використанням сучасних технологій розробки;
* тестування створеного рішення та оцінка його ефективності в умовах практичного використання;
* визначення перспектив розвитку системи.

**Практична значущість роботи** полягає у створенні вебзастосунку, який може стати базою для подальшого вдосконалення системи управління гуртожитками, включаючи інтеграцію додаткових функцій, таких як управління кількома гуртожитками, контроль за оплатою та облік майна.

### РОЗДІЛ 1

ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ПОСЕЛЕННЯ

#### 1.1. Сучасні підходи до створення програмного забезпечення для автоматизації житлових процесів

Автоматизація процесів управління житловими об'єктами стає невід'ємною складовою сучасного управлінського середовища. Зростання обсягів інформації, необхідної для обліку проживаючих, їхніх фінансових операцій, майна та інших аспектів, стимулює впровадження цифрових технологій у цій галузі. Застосування інформаційних систем дозволяє оптимізувати виконання рутинних задач, підвищити ефективність управління та зменшити ймовірність помилок, пов'язаних із людським фактором.

У сфері управління житлом активно застосовуються різні типи програмного забезпечення. Зокрема, для готелів широко використовуються системи управління готельними послугами (Hotel Management Systems, HMS), які охоплюють бронювання номерів, облік клієнтів, ведення фінансових розрахунків і управління персоналом. Такі системи, як **Opera**, **Protel**, **Cloudbeds**, демонструють високий рівень автоматизації завдяки інтеграції з іншими бізнес-системами, такими як CRM або ERP.

Для управління орендованими приміщеннями використовуються інші інструменти, наприклад, **Rentec Direct**, **Buildium** або **AppFolio**. Вони дозволяють орендодавцям та керівникам нерухомості управляти базою орендарів, контролювати фінансові потоки та забезпечувати комунікацію з клієнтами.

Проте для управління студентськими гуртожитками ситуація є дещо іншою. Зазвичай, такі об’єкти мають свої специфічні вимоги, які суттєво відрізняються від управління готелями чи орендованим житлом. Серед основних аспектів, які потребують автоматизації, виділяють:

* заселення студентів із врахуванням доступності кімнат;
* облік і контроль за оплатою проживання;
* облік майна, закріпленого за конкретними кімнатами чи мешканцями;
* можливість оперативного переселення чи виселення студентів.

Незважаючи на зростання цифровізації в цій галузі, більшість університетів досі використовують застарілі системи обліку або виконують цю роботу вручну. Це не лише знижує ефективність, а й збільшує ризик помилок у даних.

Вебзастосунки як інструмент для автоматизації облікових процесів набувають популярності завдяки своїй доступності та можливості забезпечувати швидкий доступ до даних у реальному часі. Вони дозволяють користувачам взаємодіяти з системою через браузер, що знижує витрати на інфраструктуру. Для університетів, які управляють студентськими гуртожитками, такі застосунки відкривають можливість інтеграції різних процесів в єдину платформу, спрощуючи їхнє адміністрування.

#### 1.2. Існуючі системи управління поселенням у гуртожитках

На сьогоднішній день існує ряд розробок, призначених для автоматизації процесів управління гуртожитками. Більшість із них є студентськими проєктами, створеними для локального використання в конкретних навчальних закладах. Такі системи, як правило, розробляються під час виконання курсових чи дипломних робіт і забезпечують базову функціональність для обліку мешканців і кімнат.

Серед поширених функцій таких систем можна виділити:

* створення електронної бази даних мешканців, у якій вказується основна інформація про кожного студента (ім'я, прізвище, номер кімнати);
* формування списків заселення та виселення;
* ведення журналу оплати за проживання.

Проте ці рішення, маючи обмежений функціонал, рідко використовуються на практиці. Однією з головних проблем таких систем є відсутність модулів для візуалізації даних. Університетська адміністрація змушена вручну переглядати великі списки для отримання інформації про заповненість кімнат або стан оплати мешканців. Це знижує зручність використання таких систем і впливає на їхню ефективність.

Наша система спрямована на вирішення цього недоліку шляхом інтеграції візуалізації планів поверхів із кольоровим виділенням кімнат залежно від їхнього статусу. Це дозволяє адміністрації гуртожитку швидко отримувати інформацію про:

* кількість вільних кімнат на кожному поверсі;
* кімнати, які частково заповнені;
* кімнати, що є повністю заселеними.

Така функціональність не тільки спрощує роботу адміністрації, а й дозволяє покращити планування поселення студентів. Наприклад, кімнати, які залишаються частково заселеними, можуть бути оптимально використаними для нових мешканців.

#### 1.3. Недоліки існуючих рішень та обґрунтування необхідності створення GurtKEY

Аналіз сучасних рішень для автоматизації поселення у гуртожитки виявляє кілька ключових недоліків:

* Відсутність інтерактивної візуалізації: більшість систем працюють із текстовими списками, які є складними для швидкого аналізу.
* Обмежений функціонал: часто такі системи не враховують можливість переселення мешканців, ведення обліку майна або контролю за оплатою.
* Низька інтеграція з іншими сервісами: більшість систем не підтримують можливості інтеграції з іншими програмами або базами даних університету.

Створення системи GurtKEY базується на ідеї комплексного вирішення цих проблем. Інноваційність підходу полягає у поєднанні функціоналу для автоматизації адміністративних процесів із візуалізацією даних. Основні переваги запропонованої системи:

* можливість швидкого доступу до даних завдяки графічному відображенню планів поверхів;
* підтримка таких функцій, як переселення, облік майна та контроль за оплатою;
* використання сучасних вебтехнологій, що забезпечують зручність користування і доступ із будь-якого пристрою.

Таким чином, GurtKEY не лише закриває прогалини, які існують у сучасних рішеннях, а й відкриває нові можливості для автоматизації управління гуртожитками. Цей підхід дозволяє підвищити ефективність роботи адміністрації, покращити взаємодію з мешканцями та створити більш прозору систему управління.

### РОЗДІЛ 2

МЕТОДОЛОГІЯ СТВОРЕННЯ ВЕБЗАСТОСУНКУ GURTKEY

#### 2.1 Загальні принципи розробки програмного забезпечення

Розробка сучасного програмного забезпечення базується на принципах системності, модульності, масштабованості та адаптивності. Ці принципи забезпечують надійність, зручність використання та підтримки програмних продуктів.

Основні етапи створення програмного забезпечення включають:

1. Визначення вимог. На цьому етапі розробник вивчає проблематику та визначає ключові функції, які повинен забезпечувати продукт.
2. Проектування. Розробляється архітектура системи, структурується база даних, створюються прототипи інтерфейсу.
3. Реалізація. Пишеться програмний код, інтегруються всі модулі системи.
4. Тестування. Виявляються та виправляються помилки, перевіряється функціональність.
5. Впровадження та підтримка. Система інтегрується в робоче середовище, забезпечується навчання користувачів і подальша підтримка.

Методологія розробки визначає спосіб організації роботи над проектом. Для створення вебзастосунку GurtKEY обрано метод візуального програмування, який дозволяє ефективно прототипувати інтерфейс і взаємодію користувача із системою, скорочуючи час на проектування.

#### 2.2 Метод візуального програмування

Метод візуального програмування полягає у створенні програмного забезпечення за допомогою графічних інтерфейсів та інструментів без необхідності написання коду. Одним із найбільш популярних середовищ для реалізації такого підходу є платформа Webflow.

Webflow — це інноваційна платформа для створення вебсайтів, яка забезпечує візуальний підхід до розробки інтерфейсів та дозволяє створювати складні вебпроекти без необхідності писати код. Заснована у 2013 році, Webflow швидко здобула популярність серед дизайнерів та розробників завдяки своїм потужним інструментам, що забезпечують високу функціональність та швидкість роботи. Webflow сьогодні займає вагоме місце на ринку платформ для створення сайтів, конкуруючи з такими відомими інструментами, як WordPress, Wix та Squarespace, завдяки своїй унікальній можливості поєднувати дизайн, верстку та управління контентом в одному інтерфейсі.

Для GurtKEY метод візуального програмування був обраний як інструмент для початкового проектування інтерфейсу, який згодом було вдосконалено через ручне доопрацювання коду.

#### 2.3 Використані інструменти розробки

Створення вебзастосунку **GurtKEY** виконувалося з використанням таких інструментів і технологій:

2.3.1 Webflow

Цей інструмент використовувався для розробки адаптивного інтерфейсу користувача. Webflow дозволив швидко створити макети основних сторінок і взаємодії користувача із системою.

Webflow є сучасним інструментом для розробки вебзастосунків, що надає користувачам можливості для створення адаптивного інтерфейсу користувача, автоматичної генерації HTML, CSS і базового JavaScript-коду, а також інтеграції сторонніх API через зручний інтерфейс. Цей сервіс базується на принципах візуального програмування, що дозволяє суттєво спростити процес розробки завдяки перетягуванню готових елементів та мінімізації ручного написання коду.

Метод візуального програмування, що використовується у Webflow, має низку переваг. По-перше, він забезпечує високу швидкість розробки, оскільки інтерфейси створюються за допомогою інтерактивного графічного середовища, що значно скорочує час на проектування та впровадження базового функціоналу. По-друге, цей метод є інтуїтивно зрозумілим, що дозволяє навіть користувачам із базовими знаннями у програмуванні розробляти складні вебзастосунки. По-третє, висока продуктивність на етапі тестування досягається завдяки можливості виявлення помилок ще під час побудови структури інтерфейсу, що значно зменшує витрати часу на їх усунення у пізніших стадіях.

Попри це, метод візуального програмування має й певні недоліки. До них належить обмеженість функціоналу у порівнянні з ручним програмуванням, що може створювати труднощі при реалізації специфічних або складних вимог. Крім того, існує залежність від платформи, що може обмежувати користувача у виборі середовища розробки та викликати труднощі при перенесенні проєкту до інших систем.

* + 1. ExFlow

Після створення макету інтерфейсу в Webflow був застосований парсер, який дозволив конвертувати отриманий код для подальшого використання у VSC. ExFlow — це інструмент, який дозволяє повністю експортувати сайт, створений на платформі Webflow, включаючи сторінки з динамічним контентом CMS. Це важлива функціональність, яка відсутня навіть у платних тарифних планах Webflow. ExFlow створено для дизайнерів і розробників, які використовують Webflow для створення сайтів, але хочуть розміщувати їх на сторонніх серверах або інтегрувати в інші екосистеми. Інструмент дозволяє налаштувати структуру експортованих даних, задаючи імена папок для оптимальної організації файлів. ExFlow доступний безкоштовно, що робить його економічно вигідним рішенням.

Правові аспекти використання ExFlow:

Згідно з міжнародними законами про інтелектуальну власність, зокрема Бернською конвенцією, автор зберігає права на створений контент. Якщо сайт розроблений самостійно, права на його дизайн та контент належать користувачеві.

Умови обслуговування (Terms of Service, TOS) Webflow не забороняють експорт оригінального контенту користувача, якщо він належить автору. Однак слід уважно ознайомлюватися з актуальними положеннями TOS, оскільки вони можуть змінюватися.

* + 1. Visual Studio Code (VSC)

Після створення базового коду за допомогою Webflow, він був експортований і вдосконалений у редакторі Visual Studio Code (VSC). Visual Studio Code є безкоштовним текстовим редактором із відкритим вихідним кодом, розробленим компанією Microsoft, який широко використовується для програмування завдяки своїй багатофункціональності, простоті у використанні та підтримці численних мов програмування. Будучи кросплатформним інструментом, VSC працює на операційних системах Windows, macOS і Linux, що забезпечує його універсальність і зручність для розробників.

Visual Studio Code має низку ключових функцій, які роблять його незамінним для сучасного програмування. Зокрема, редактор підтримує понад 30 мов програмування «з коробки», включаючи JavaScript, Python, C++, PHP, HTML і CSS, а також безліч інших мов через додаткові розширення, доступні в маркетплейсі. Інтеграція з Git дозволяє розробникам відстежувати зміни у файлах, виконувати коміти, створювати та зливати гілки, а також працювати з віддаленими репозиторіями безпосередньо з редактора.

Маркетплейс розширень пропонує тисячі додаткових модулів, які додають нові функції, такі як автодоповнення коду для специфічних мов, лінтери для перевірки синтаксису та інтеграція із зовнішніми системами, такими як Docker або AWS. Інтелектуальні функції для кодування, зокрема IntelliSense, забезпечують розширене автозаповнення коду з підказками щодо змінних, функцій і параметрів. Додатково VSC пропонує виділення синтаксису та автоматичне форматування, що робить процес роботи з кодом ефективнішим і зручнішим.

Для налагодження програм VSC має вбудований дебагер, який дозволяє виконувати покроковий аналіз програм і знаходити помилки без необхідності використання зовнішніх засобів. Крім того, вбудований термінал дає змогу виконувати команди без переключення між редактором і зовнішньою консоллю.

У рамках удосконалення коду вебзастосунку GurtKEY за допомогою Visual Studio Code було реалізовано кілька важливих аспектів. Зокрема, оптимізовано структуру HTML-коду для підвищення його читабельності та зручності підтримки, додано нові функції за допомогою JavaScript і виконано інтеграцію з базою даних для забезпечення динамічної взаємодії користувачів із системою.

* + 1. Postman

Postman є багатофункціональною платформою, призначеною для роботи з API (інтерфейсами прикладного програмування), що охоплює весь цикл їх розробки, тестування, документування та моніторингу. Цей інструмент широко використовується для роботи з RESTful API, а також підтримує інші протоколи, такі як SOAP і GraphQL, забезпечуючи розробникам гнучкість і зручність на всіх етапах роботи.

Однією з основних функцій Postman є створення та виконання API-запитів. Інструмент підтримує всі основні HTTP-методи, включаючи GET, POST, PUT, DELETE та PATCH, і дозволяє задавати параметри URL, заголовки, а також тіло запиту у форматах JSON, XML чи форм-даних. Завдяки цьому розробники мають змогу точно налаштовувати запити відповідно до вимог API.

Платформа також забезпечує ефективне тестування API. Вбудований функціонал дозволяє автоматизувати перевірки працездатності API, створюючи скрипти на JavaScript для перевірки статус-кодів, відповідності структури JSON-відповідей, часу відповіді та інших параметрів. Це дозволяє швидко ідентифікувати проблеми та підвищує якість роботи API.

Postman надає можливість об’єднувати кілька запитів у колекції, які можна використовувати для тестування, демонстрації чи спільної роботи над API. Колекції можна експортувати, що значно спрощує обмін і повторне використання налаштувань між командами розробників.

Інструмент також підтримує моніторинг API, що дозволяє регулярно перевіряти його доступність і продуктивність. Це важливо для забезпечення стабільної роботи API в реальних умовах. Завдяки інтеграції з платформами CI/CD, такими як Jenkins, GitHub Actions чи Azure DevOps, Postman дозволяє автоматизувати перевірки API у процесі розробки, що сприяє безперервному вдосконаленню продукту.

Однією з важливих функцій Postman є генерація документації API. Платформа автоматично створює детальну документацію на основі запитів, що використовуються в ній, і дозволяє публікувати її в хмарному середовищі. Ця можливість забезпечує прозорість і зручність використання API для розробників і зацікавлених сторін.

Postman також інтегрується з іншими інструментами, такими як Swagger, GitHub, Slack, що розширює його функціональність і спрощує спільну роботу над API.

У рамках роботи над проєктом Postman використовувався для тестування API-запитів до бази даних. Це дозволило перевірити коректність отримання та запису даних, забезпечивши функціональність системи в реальних умовах і гарантуючи її надійність у процесі експлуатації.

#### 2.4 Архітектура системи

Архітектура системи **GurtKEY** базується на моделі **клієнт-сервер**. Клієнт-серверна модель — це архітектурний підхід до побудови інформаційних систем, який передбачає розподіл функцій між двома типами програм: **клієнтом** і **сервером**. У цій моделі клієнт ініціює запити, а сервер обробляє їх і повертає відповідь.

Ця модель широко використовується в комп'ютерних мережах, особливо в контексті Інтернету, де сервери забезпечують доступ до вебсайтів, баз даних, файлів або інших ресурсів, а клієнти — це пристрої чи програми, які отримують ці ресурси.

Модель клієнт-сервер забезпечує розподіл обчислювальних навантажень між клієнтською і серверною частинами, що сприяє підвищенню продуктивності та масштабованості системи.

#### 2.5 Етапи створення GurtKEY

2.5.1 Аналіз вимог

Розробка програмного забезпечення для обліку поселення в гуртожитках вимагає чіткого визначення функціональних і нефункціональних вимог, а також аналізу потреб майбутніх користувачів. В основу програмного забезпечення покладено кілька ключових завдань, зокрема автоматизацію процесу поселення мешканців, забезпечення обліку проживаючих, контроль за оплатою та майном у кімнатах. Основні вимоги були сформульовані на основі аналізу типових сценаріїв використання таких систем.

Програмне забезпечення має бути сумісним із платформою Windows, починаючи з версії XP, хоча оптимальним варіантом є забезпечення кросплатформності для ширшого охоплення користувачів. Головною особливістю програми є візуалізація стану поселення за допомогою інтерактивного плану поверхів, який відображає статус кімнат за допомогою кольорової індикації: зеленим кольором позначаються вільні кімнати, жовтим — частково заселені, а червоним — повністю заселені. Окрім цього, програма повинна забезпечувати можливість редагування планів поверхів, зміну кількості місць у кімнатах і управління даними про мешканців.

Система орієнтована на два основні типи користувачів: «Поселення» і «Комендант гуртожитка». Для користувача «Поселення» передбачено такі функції, як візуалізація стану поселення, введення даних про мешканців (наприклад, їхнє ім’я, факультет, на якому вони навчаються, контакти тощо), виконання операцій із поселення, переселення, виселення мешканців і ведення історії використання кімнат. Ці функції дозволяють максимально автоматизувати процеси, пов’язані з поселенням, і забезпечити їхню простоту для кінцевих користувачів.

Для користувача «Комендант гуртожитка» передбачено розширений функціонал. До його основних обов’язків належить контроль за обліком мешканців, моніторинг стану кімнат, ведення історії поселення, а також облік і контроль за оплатою мешканців. Останнє включає відстеження заборгованості та формування відповідних звітів. Крім цього, комендант має можливість обліку майна в кімнатах: фіксації меблів, техніки та іншого обладнання, а також записів про їх переміщення, списання чи оновлення.

Інтерфейс програми має бути простим і зручним для користувачів, забезпечуючи легкий доступ до всіх функцій. Основний акцент робиться на інтерактивному плані поверхів, який дозволяє швидко отримувати інформацію про стан кімнат і виконувати основні операції. Важливим є також забезпечення можливості введення і редагування даних через зручні форми, що мінімізує час роботи користувачів із системою.

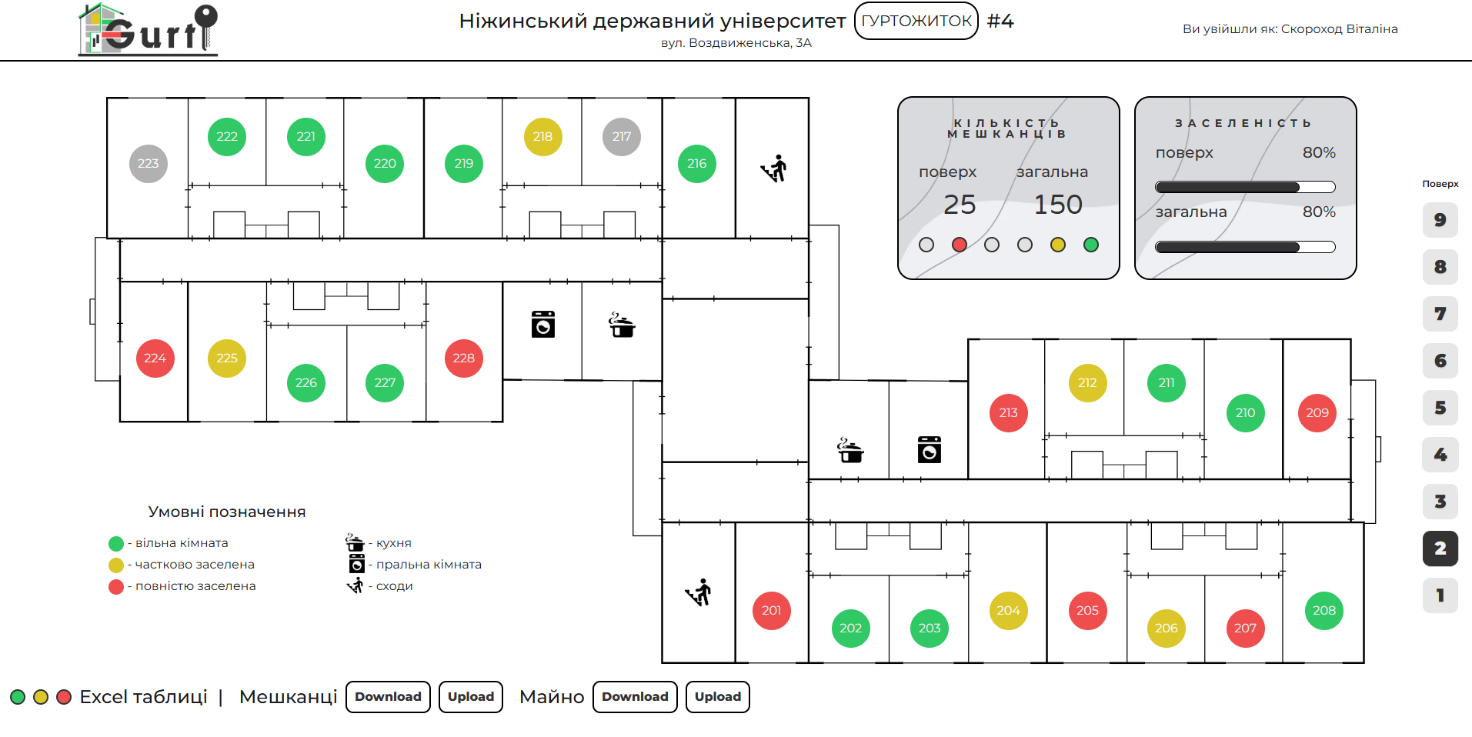
У перспективі передбачено можливість розширення функціоналу системи. Зокрема, це може включати автоматичну генерацію звітів про стан поселення, облік оплати, а також створення особистих кабінетів для мешканців. Така функціональність дозволить кожному мешканцю отримати доступ до своїх даних і статусу оплати. Крім того, доцільним буде розгляд інтеграції програми з базами даних університетів або інших організацій для автоматичного імпорту даних про студентів чи працівників.

Серед основних обмежень слід зазначити вимогу підтримки старих версій операційних систем, таких як Windows XP, що може ускладнити використання сучасних технологій. Водночас значний обсяг даних і висока інтерактивність системи потребують оптимізації алгоритмів для забезпечення стабільної роботи.

Таким чином, розробка даного програмного забезпечення покликана забезпечити автоматизацію процесів управління поселенням у гуртожитках. Вона базується на врахуванні потреб основних користувачів, інтуїтивності інтерфейсу, масштабованості та можливості подальшого розширення функціоналу.

2.5.2 Проектування інтерфейсу

Проектування інтерфейсу для системи обліку поселення в гуртожиток є важливим етапом розробки, що забезпечує ефективну взаємодію користувачів із програмним забезпеченням. Основними принципами, яких дотримувалися під час створення інтерфейсу, стали простота, інтуїтивність, доступність та адаптивність.

Рисунок 1. Інтерфейс веб застосунку GurtKEY

Інтерфейс системи розроблено з урахуванням зручності використання та інтуїтивної взаємодії користувачів із ключовими компонентами. Він включає в себе кілька основних елементів, які забезпечують ефективне управління даними та швидкий доступ до необхідної інформації.

Центральним елементом інтерфейсу є візуальна карта поверху гуртожитку. Вона являє собою інтерактивний план, що відображає стан кожної кімнати. Такий підхід дозволяє користувачам одразу оцінити завантаженість кімнат, визначити їх статус (вільна, частково зайнята, повністю зайнята) та швидко знайти потрібну кімнату. Клікабельність кожної кімнати забезпечує перехід до детальної інформації, яка включає дані про мешканців і стан приміщення, що сприяє спрощенню навігації та оперативності прийняття рішень.

Додатковим важливим компонентом інтерфейсу є інформаційна панель, розташована у правій частині екрана. Вона надає ключову статистичну інформацію, яка включає кількість мешканців, що проживають на поточному поверсі та в гуртожитку загалом. Крім того, панель відображає відсоток заповнення кімнат, дозволяючи порівнювати заповненість окремого поверху та всього гуртожитку. Такий підхід забезпечує візуалізацію важливих даних у зручному форматі, що сприяє підвищенню ефективності роботи адміністрації гуртожитку та прийняттю оперативних управлінських рішень.

Таким чином, інтеграція візуальної карти поверху та інформаційної панелі забезпечує зручність, наочність і функціональність інтерфейсу системи, відповідаючи сучасним вимогам до автоматизації процесів управління гуртожитками.

Інтерфейс системи включає додаткові елементи, які забезпечують гнучкість у взаємодії з даними та дозволяють ефективно виконувати адміністративні завдання. Одним із таких елементів є панель показників, яка надає ключові метрики у вигляді числових значень і графічних індикаторів. Цей компонент дає можливість швидко оцінити стан заповненості гуртожитку, що є особливо корисним для комендантів у процесі моніторингу поточного стану проживання.

Для управління кімнатами та мешканцями передбачено використання модальних вікон. Ці вікна дозволяють адміністраторам переглядати, додавати, редагувати або видаляти інформацію про мешканців конкретної кімнати, а також вести облік майна. Такий підхід забезпечує централізовану взаємодію з даними та підвищує ефективність управління.

Система також підтримує функції експорту та імпорту даних. У нижній частині інтерфейсу реалізовано можливість експортувати дані у форматі таблиць Excel, що спрощує створення звітів і обмін інформацією. Крім того, доступна опція завантаження наявних даних у систему, що дозволяє легко переносити інформацію між різними платформами та забезпечує інтеграцію із зовнішніми системами.

Одним із ключових елементів є картка кімнати, яка відкривається у вигляді модального вікна при виборі конкретної кімнати. Ця картка відображає список мешканців і дані про зайнятість місць, забезпечуючи зручний доступ до актуальної інформації. Адміністратор має можливість додавати нового мешканця, редагувати наявні записи або видаляти дані, а також управляти майном кімнати.

Дизайн інтерфейсу розроблено на основі принципів мінімалізму, інтуїтивності та зрозумілості. Використання кольорового кодування дозволяє легко ідентифікувати стан кімнат завдяки чіткій візуалізації даних, що сприяє швидкому аналізу. Застосування мінімалістичного підходу зменшує кількість зайвих елементів, забезпечуючи фокус користувача на виконанні основних завдань. Інтуїтивна взаємодія досягається через використання зрозумілих іконок та підказок, що робить систему доступною навіть для нових користувачів.

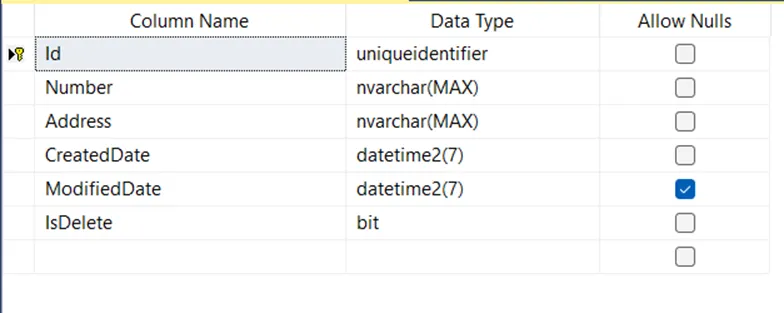
Проектування інтерфейсу системи спрямоване на забезпечення максимальної зручності для адміністрації гуртожитку. Інтерфейс дозволяє ефективно управляти процесом поселення, отримувати повну інформацію про мешканців і підтримувати актуальність даних. Завдяки використанню інтерактивних елементів, система є простою у використанні та відповідає сучасним вимогам до програмного забезпечення.

2.5.3 Розробка бази даних

Для реалізації функціоналу вебзастосунку GurtKEY було спроектовано структуру бази даних із таблицями, призначеними для обліку кімнат, мешканців, оплати та майна. Розробка бази даних виконувалася за методом **code-first**, де початково створювалися сутності у вигляді класів у програмному коді, після чого здійснювалася їх міграція в базу даних.

Підключення до бази даних виконувалося за допомогою SQL Server Management Studio (SSMS). Для перегляду структури таблиць використовувалася функція **Design**, яка дозволяє досліджувати властивості таблиць, їхні поля та зв’язки між ними. Основні таблиці бази даних включають: **Hostels**, **Persons** та **Rooms**.

Таблиця Hostels

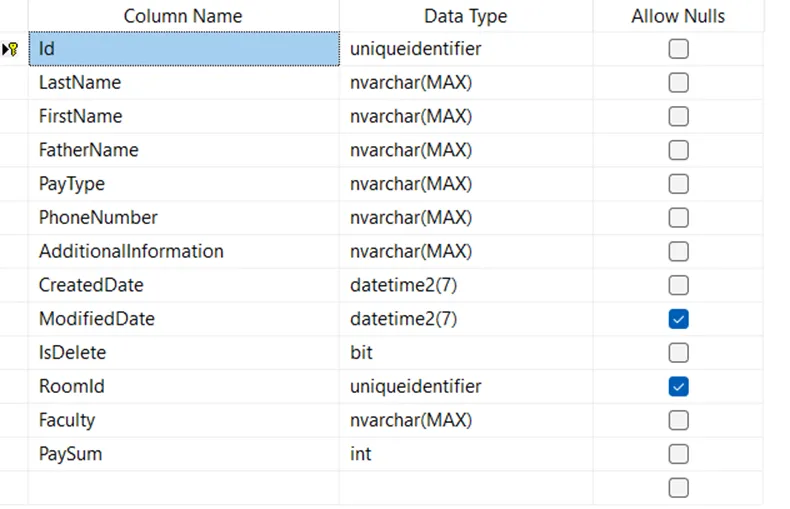
Рисунок 2. Структура таблиці Hostels

Таблиця **Hostels** містить інформацію про гуртожитки і включає шість полів:

* **Id** – унікальний ідентифікатор запису, який є первинним ключем (Primary Key), формату **Guid**.
* **Number** – номер гуртожитку, типу **nvarchar**.
* **Address** – адреса гуртожитку, типу **nvarchar**.
* **CreatedDate** – дата створення запису, типу **datetime**.
* **ModifiedDate** – дата редагування запису, типу **datetime**, значення може бути відсутнім (**nullable**).
* **IsDelete** – ознака м’якого видалення (**soft delete**), типу **bool**.

Механізм м’якого видалення передбачає зміну значення поля **IsDelete** на true замість фізичного видалення запису, що дозволяє зберігати історичні дані в базі.

Таблиця Persons

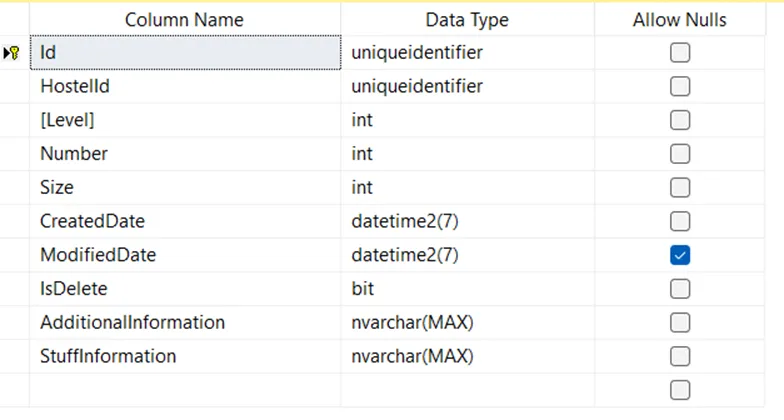
Рисунок 3. Структура таблиці Persons

Таблиця **Persons** містить дані про мешканців кімнат і включає наступні поля:

* **Id** – унікальний ідентифікатор запису, первинний ключ (**Guid**).
* **LastName**, **FirstName**, **FatherName** – прізвище, ім’я та по батькові мешканця, типу **nvarchar**.
* **PayType** – спосіб оплати, типу **nvarchar**.
* **PhoneNumber** – номер телефону, типу **nvarchar**.
* **AdditionalInformation** – додаткова інформація або коментар, типу **nvarchar**.
* **CreatedDate** – дата створення запису, типу **datetime**.
* **ModifiedDate** – дата редагування запису, типу **datetime**, значення може бути відсутнім.
* **IsDelete** – ознака м’якого видалення, типу **bool**.
* **RoomId** – ідентифікатор кімнати, формату **Guid**, значення може бути відсутнім.
* **Faculty** – назва факультету, типу **nvarchar**.
* **PaySum** – сума оплати, типу **integer**.

Ця таблиця дозволяє зберігати детальну інформацію про кожного мешканця, включаючи фінансові дані та додаткові характеристики.

Таблиця Rooms

Рисунок 4. Структура таблиці Rooms

Таблиця **Rooms** містить дані про кімнати у гуртожитках і має такі поля:

* **Id** – унікальний ідентифікатор запису, первинний ключ (**Guid**).
* **HostelId** – ідентифікатор гуртожитку, до якого належить кімната, типу **Guid**.
* **Level** – номер поверху, типу **integer**.
* **Number** – номер кімнати, типу **integer**.
* **Size** – кількість місць у кімнаті, типу **integer**.
* **CreatedDate** – дата створення запису, типу **datetime**.
* **ModifiedDate** – дата редагування запису, типу **datetime**, значення може бути відсутнім.
* **IsDelete** – ознака м’якого видалення, типу **bool**.
* **AdditionalInformation** – додаткова інформація, типу **nvarchar**.
* **StuffInformation** – інформація про майно у кімнаті, типу **nvarchar**.

Таблиця **Rooms** забезпечує зберігання даних про приміщення, їхній статус і характеристики, що є основою для обліку мешканців та управління майном.

Створена структура бази даних із таблицями **Hostels**, **Persons** та **Rooms** забезпечує всі необхідні умови для реалізації функціоналу обліку поселення у гуртожитках. Завдяки використанню методу **code-first** структура є добре організованою та адаптованою до подальшого розширення. Така архітектура бази даних гарантує ефективну роботу системи в реальних умовах.

2.5.4 Інтеграція клієнтської та серверної частин

Взаємодія між інтерфейсом користувача та базою даних у системі GurtKEY забезпечується за допомогою **API (інтерфейсу прикладного програмування)**, що слугує проміжним рівнем для обміну даними. Цей підхід гарантує ефективність, безпеку та структурованість передачі інформації між фронтендом і бекендом системи.

Архітектура взаємодії

1. **Запити з інтерфейсу користувача. І**нтерфейс користувача, розроблений на основі HTML, CSS та JavaScript, генерує запити до API для виконання операцій, таких як отримання списків кімнат, додавання мешканців чи оновлення інформації. Запити відправляються через асинхронні HTTP-запити за допомогою функцій fetch або бібліотек, що працюють із RESTful API.
2. **API як посередник.** API є проміжним рівнем між інтерфейсом користувача та базою даних. Воно приймає запити від фронтенда, обробляє їх і виконує відповідні дії над базою даних. API побудоване на RESTful архітектурі, що дозволяє виконувати CRUD-операції (створення, читання, оновлення, видалення) через HTTP-методи:
   * GET — для отримання даних із бази (наприклад, списків кімнат чи інформації про мешканців).
   * POST — для створення нових записів у базі (наприклад, додавання нового мешканця).
   * PUT/PATCH — для оновлення існуючих записів (наприклад, редагування інформації про кімнату).
   * DELETE — для видалення записів (реалізується через soft delete у системі).
3. **Запити до бази даних.** Після отримання запиту API виконує запит до бази даних за допомогою ORM (об’єктно-реляційного відображення), реалізованого в бекенді. У цьому проєкті було застосовано підхід **code-first**, що дозволяє API працювати зі структурою бази даних через сутності, представлені у вигляді класів. ORM автоматично трансформує запити до сутностей у SQL-запити до бази даних.
4. **Обробка даних і передача відповідей.** Після отримання даних із бази вони обробляються на рівні API, а результат передається у форматі JSON назад до інтерфейсу користувача. JSON-формат забезпечує легкість обробки даних у фронтенді, дозволяючи динамічно оновлювати інтерфейс користувача без перезавантаження сторінки.
5. **Інтерактивність інтерфейсу.** Отримані дані інтегруються в інтерфейс користувача. Наприклад:
   * Відображення списку мешканців у картці кімнати.
   * Оновлення візуального стану кімнат після змін у базі.
   * Додавання чи редагування інформації, введеної користувачем, через інтерактивні форми.

Забезпечення безпеки та ефективності

* **Авторизація**: Усі запити до API містять API-ключ для підтвердження доступу. Це захищає систему від несанкціонованих запитів.
* **Обробка помилок**: API передає користувачеві повідомлення про статус операції (успішне виконання, помилка).
* **Кешування даних**: Для зменшення навантаження на сервер можуть використовуватися механізми кешування часто використовуваних даних.

Таким чином, взаємодія між інтерфейсом користувача та базою даних реалізована через API, що забезпечує ефективну, безпечну та надійну роботу системи GurtKEY у реальних умовах.

Таким чином, система GurtKEY була спроєктована з використанням сучасних методів та інструментів, що забезпечують її зручність, надійність і можливість подальшого розширення.

### РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ GURTKEY

3.1 Верстка інтерфейсу з Webflow

У сучасному процесі розробки вебзастосунків одним із ключових аспектів є створення зручного та інтуїтивно зрозумілого інтерфейсу користувача. Інтерфейс виконує роль посередника між системою та користувачем, забезпечуючи ефективну взаємодію та доступ до функціональних можливостей програмного забезпечення. Саме тому важливим завданням є не лише розробка візуально привабливого дизайну, але й забезпечення його адаптивності, логічної структури та функціональності.

У даному розділі буде представлено процес реалізації основних елементів інтерфейсу користувача для вебзастосунку GurtKEY із використанням платформи Webflow. Розглядаються механізми створення ключових компонентів, зокрема панелі навігації, плану приміщення, візуальних індикаторів та інформаційних блоків. Також демонструються способи застосування вбудованих інструментів анімації та стилізації, які дозволяють забезпечити високий рівень зручності та естетики інтерфейсу.

Мета цього розділу полягає у висвітленні практичних аспектів розробки інтерфейсу з використанням Webflow, а також у поясненні прийнятих рішень щодо дизайну та юзабіліті. Представлені методики та підходи можуть бути корисними для подальшого вдосконалення системи та слугуватимуть прикладом ефективної реалізації користувацьких інтерфейсів.

Інтерфейс системи було розроблено з акцентом на зручність і простоту використання.

Основні компоненти інтерфейсу:

3.1.1 Навігаційна панель

Навігаційна панель (англ. navigation bar, navbar) — це елемент графічного інтерфейсу користувача, який призначений для забезпечення зручного та швидкого доступу до основних розділів, сторінок або функцій вебзастосунку чи сайту. Вона виконує роль орієнтира, допомагаючи користувачам зрозуміти структуру ресурсу та швидко переходити до потрібної інформації.

У нашому випадку, де сайт розроблено у форматі односторінкового лендінгу, навігаційна панель виконує роль не лише засобу навігації, але й інформаційної панелі. Основний акцент у її структурі зроблено на представлення загальної інформації про гуртожиток, яка займає більшу частину доступного простору.

До основних елементів панелі відносяться:

* Логотип системи, який візуалізує її ідентичність та дозволяє користувачам легко впізнавати ресурс. Його створено за допомогою div-блоку, який містить зображення логотипу. Під зображенням логотипу інтегровано тригер для відкриття випадаючого меню, що дозволяє обирати доступні гуртожитки. Відкриття та закриття випадаючого меню реалізовано за допомогою вбудованого редактора анімацій у Webflow.
* Назва гуртожитку розташована в центральній частині панелі навігації. Цей елемент відображає номер гуртожитку, в якому наразі перебуває користувач. Він оформлений як текстовий блок із використанням відповідних стилів для вирівнювання та візуального акценту. Під назвою гуртожитку розміщено текстовий елемент, що містить адресу. Це додатковий текстовий блок, що забезпечує інформативність.

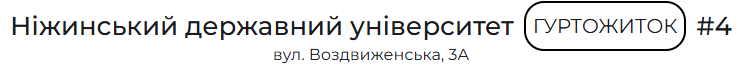


Рисунок 5. Назва гуртожитку в елементі header

* Інформація про поточного користувача. У правій частині навігаційної панелі знаходиться текстовий блок, який відображає ім’я користувача, який увійшов в систему. Цей блок також може містити додаткові функції, такі як меню профілю, хоча наразі реалізовано лише текстову інформацію.
* Механізм перемикання між гуртожитками, реалізований через випадаюче меню, що забезпечує швидкий доступ до даних інших об'єктів. У структурі навігаційної панелі створено блок із класом dropdown content, який містить список доступних гуртожитків. Кожен пункт списку реалізовано як окремий елемент dropdown element, що забезпечує швидкий доступ до вибору конкретного об’єкта. Відповідне меню випадає при натисканні на логотип сайту.

Для всіх елементів панелі навігації застосовано макет flexbox, який забезпечує рівномірний розподіл елементів по горизонталі та дозволяє налаштовувати вирівнювання залежно від розмірів екрана. Для кожного блоку та текстового елемента налаштовано відступи (padding і margin) у редакторі стилів, що дозволяє досягти візуальної гармонії та оптимального користувацького досвіду.

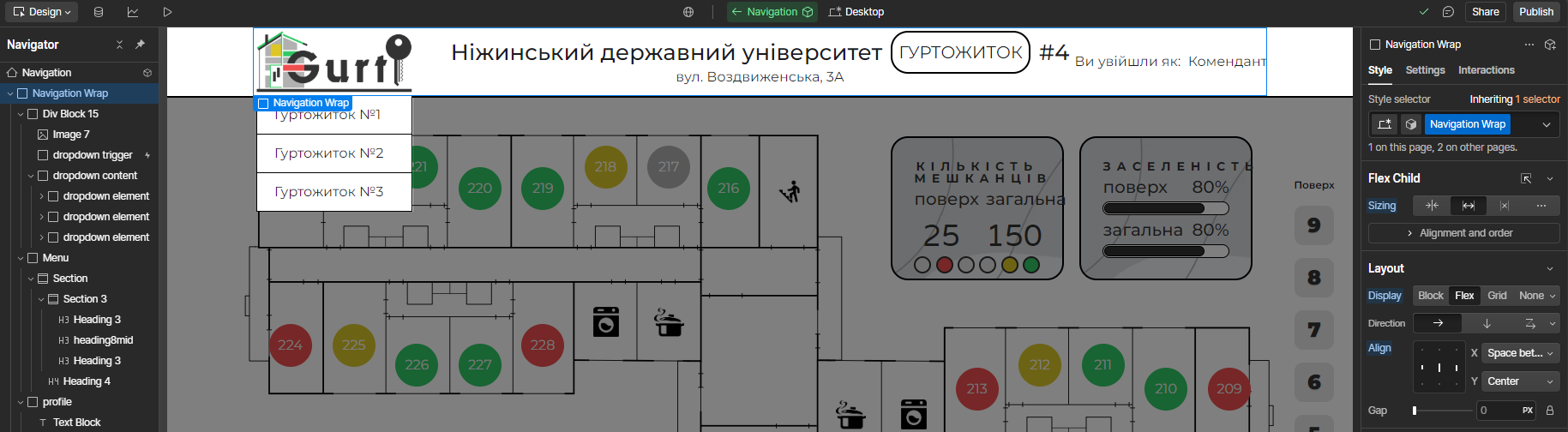


Рисунок 6. Інтерфейс веб застосунку Webflow

Усі елементи панелі навігації оптимізовано для роботи на різних пристроях. Випадаюче меню та текстові блоки зберігають функціональність та естетичність під час зміни розмірів вікна браузера.

3.1.2 Розташування та функціональність елементів сторінки

Основну частину сторінки займає інтерактивний план гуртожитку, що є візуальним представленням розташування кімнат на поверсі. План створено у векторному форматі, що забезпечує його масштабованість без втрати якості, навіть при зміні розмірів вікна браузера. Такий підхід дозволяє користувачам комфортно взаємодіяти з елементами інтерфейсу незалежно від роздільної здатності пристрою.

Для інтерактивності на зображення було накладено grid-контейнер, у якому виділено 36 спеціальних комірок (*area*), що відповідають розташуванню кімнат. Кожна комірка займає певну область, чітко визначену у grid-сітці, і слугує основою для розміщення кнопок-кімнат. Завдяки структурі grid-контейнера забезпечується:

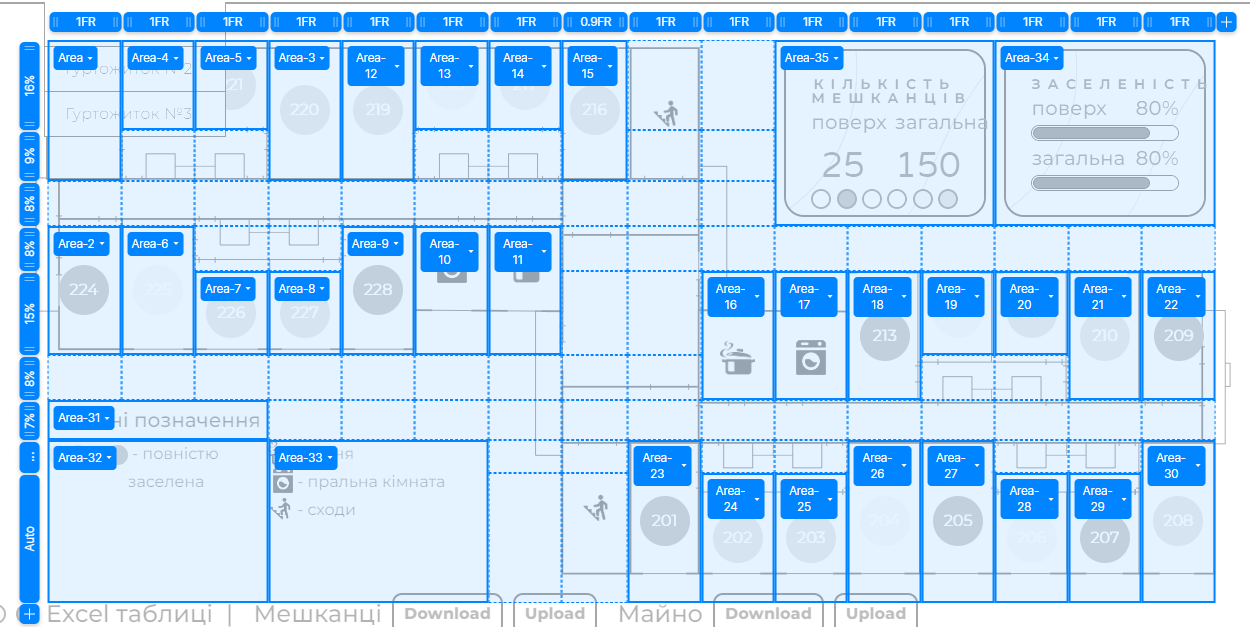
* Точність позиціонування: Кнопки чітко відповідають позиції кімнат на плані.
* Гнучкість адаптації: Можливість швидкої модифікації розташування елементів у разі змін планування або дизайну.

Рисунок 7. Grid-контейнер

Кожна кнопка кімнати представлена у вигляді div-блоку із такими характеристиками:

* Форма: Кругла, з радіусом 20 пікселів.
* Кольорове кодування: Для позначення стану кімнат створено чотири класи, кожен з яких має свій колір:
  + Зелений (#31c966) — кімната повністю заселена.
  + Жовтий (#dbc72a) — кімната частково заселена.
  + Червоний (#ee4e4e) — кімната недоступна.
  + Сірий (#b1b1b1) — кімната неактивна або закрита.
* Колір тексту: Білий, для забезпечення контрастності та читабельності.

При натисканні на будь-яку з кнопок відкривається картка з детальною інформацією про обрану кімнату, що дозволяє користувачеві отримати актуальні дані про її стан.

У правому верхньому куті сторінки розміщено картки, що відображають ключові статистичні дані про гуртожиток та поверх. Картки розташовані в окремих контейнерах, які вирівняні по центру горизонтальної осі, забезпечуючи симетричність і зручне сприйняття інформації. Кожен контейнер має відступ у 4 пікселі від усіх країв, що створює візуально приємний простір між елементами. Перша картка містить інформацію про кількість мешканців поверху та загальну кількість мешканців гуртожитку, розташовану у двох колонках: числові показники та індикатори станів кімнат. Друга картка ілюструє рівень заселеності за допомогою горизонтальних шкал, де ширина внутрішнього елемента шкали відповідає відсотковому значенню. Така організація дозволяє швидко отримувати необхідну інформацію, а структуроване розташування елементів сприяє зручності використання.

Рисунок 8. Картки зі статистикою

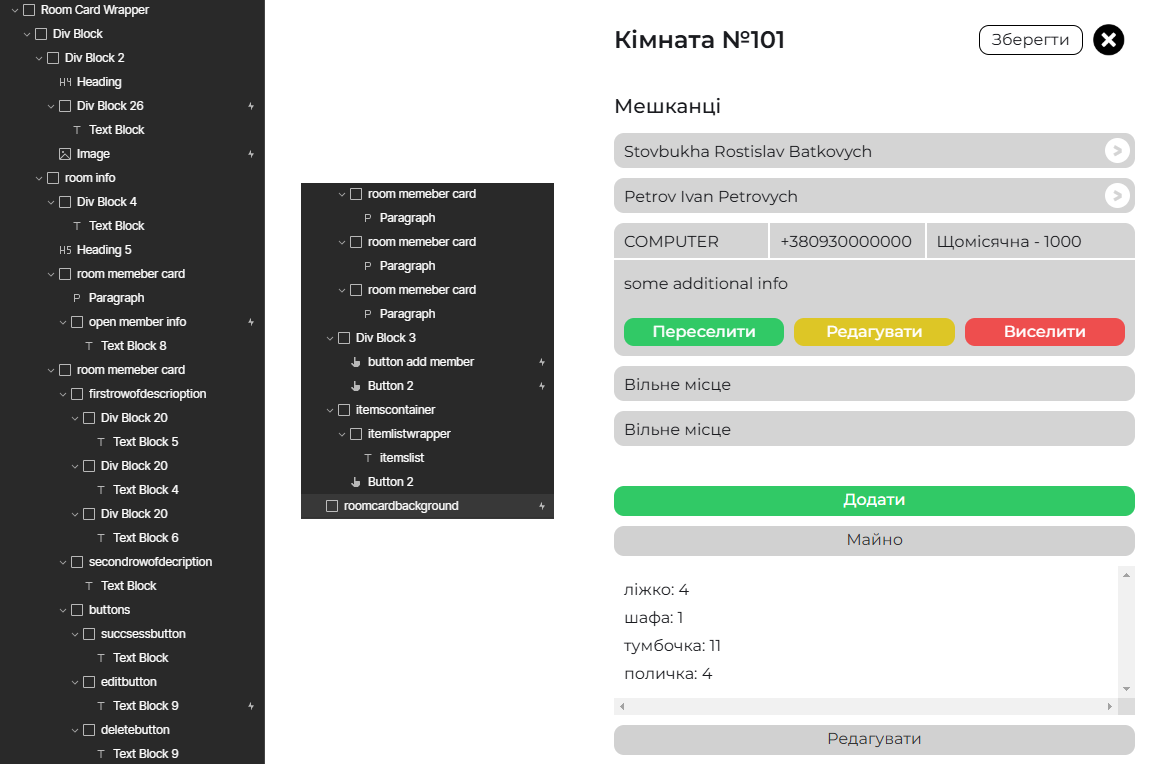
У лівому нижньому куті розташована легенда, яка пояснює кольорове кодування кнопок-кімнат та позначення інших елементів плану. Легенда забезпечує зрозумілість для користувачів і спрощує орієнтацію на сторінці.

У правій частині сторінки розташовано вертикальний список кнопок для перемикання між поверхами. Кожна кнопка реалізована у вигляді div-блоку, колір фону —­ Світло-сірий (#e7e7e7), радіус — 10px. І містить у собі текстовий блок з номером поверху, колір тексту — темно-сірий (#333333), висота шрифту — 24px.

Для активного поверху створено окремий клас floorbuttonactive, що змінює кольорову палітру (інверсія кольорів), що допомагає візуально ідентифікувати обраний поверх.

3.1.3 Картка кімнати

Картка кімнати у вебзастосунку є ключовим елементом інтерфейсу, який надає користувачеві можливість взаємодії з даними про конкретне приміщення.

Рисунок 9. Картка кімнати

У верхній частині картки розташовано заголовок із номером кімнати, а також кнопки "Зберегти" для підтвердження змін і "Закрити" для виходу з цього інтерфейсу. Під заголовком розміщено блок із загальною інформацією про кімнату. Основну частину картки займає секція "Мешканці", яка містить картки з даними про кожного мешканця, такими як прізвище, ім'я, номер телефону та додаткова інформація.

Для управління статусом мешканців передбачено кнопки дій: "Переселити", "Редагувати" та "Виселити", які мають відповідне кольорове кодування (зелений, жовтий, червоний). Нижче розташовані порожні місця для додавання нових мешканців і кнопка "Додати", яка дозволяє здійснити цю дію. Наприкінці картки розміщено секцію "Майно", що відображає інформацію про наявність та кількість предметів меблів у кімнаті, таких як ліжка, шафи, тумбочки та полиці. Усі елементи картки організовані в окремих контейнерах із рівномірними відступами.

У цьому, було проаналізовано реалізацію ключових елементів дизайну, які забезпечують зручність та функціональність роботи системи. Було створено основні компоненти сторінки, включно з навігаційною панеллю, інтерактивним планом гуртожитку, картками статистики та деталізованими картками кімнат. Описані механізми організації цих елементів, їхні стилістичні та функціональні особливості, зокрема використання кольорового кодування, адаптивних сіток і інтерактивних кнопок.

Реалізація візуальних компонентів була спрямована на досягнення інтуїтивно зрозумілого інтерфейсу, який відповідає сучасним принципам вебдизайну. Усі елементи ретельно спроектовані для забезпечення зручної взаємодії користувача із системою, мінімізації можливих помилок і підвищення ефективності роботи. Представлений підхід до візуалізації демонструє, як гармонійне поєднання функціональності, адаптивності та естетики сприяє створенню якісного програмного продукту, який відповідає потребам користувачів і специфіці завдань.

3.2 Взаємодія з базою даних

Серверна частина використовує ORM (об’єктно-реляційне відображення) для зручної роботи з реляційною базою даних. Завдяки ORM запити до бази формуються на основі сутностей у коді, що дозволяє автоматизувати багато процесів і зменшити кількість ручного написання SQL-запитів.

База даних побудована за реляційним принципом, що забезпечує структуроване зберігання даних і можливість створення зв’язків між таблицями. Вона складається з таблиць для обліку мешканців, кімнат, оплати та майна.

Для забезпечення ефективної роботи бази даних застосовано такі підходи:

Оптимізація запитів:

* Індексація: Для ключових полів (наприклад, Id, RoomId, HostelId) створено індекси, що дозволяє значно прискорити виконання пошукових і фільтраційних запитів.
* Кешування: Часто використовувані дані, такі як список кімнат чи мешканців, можуть зберігатися у кеші для зменшення навантаження на сервер і прискорення роботи.
* Пагінація: Для великих наборів даних реалізовано розбиття результатів на сторінки, що дозволяє уникнути перевантаження серверних ресурсів.

Забезпечення цілісності даних:

* Ключові обмеження: Первинні ключі (Primary Key) забезпечують унікальність записів, а зовнішні ключі (Foreign Key) гарантують зв’язки між таблицями (наприклад, між кімнатами та мешканцями).
* Валідація на рівні бази: Встановлено обмеження для обов’язкових полів (наприклад, RoomId у таблиці мешканців не може бути порожнім для заселених мешканців).
* Soft delete: Для уникнення втрати даних використовується м’яке видалення, що дозволяє позначати записи як видалені без їхнього фізичного вилучення з бази.

Таким чином, серверна частина та база даних працюють у тісному взаємозв’язку, забезпечуючи коректну обробку запитів, перевірку введених даних і надійне зберігання інформації.

3.3 Реалізація основних функцій

У цьому розділі буде розглянуто технічну реалізацію функціоналу вебзастосунку, зокрема його структуру та ключові аспекти розробки за допомогою HTML і JavaScript. На цьому етапі створення програмного продукту головною метою є забезпечення чіткої взаємодії між візуальними елементами інтерфейсу та їхньою функціональною складовою. HTML використовується для побудови основної структури сторінок, а JavaScript виконує роль мови програмування, що дозволяє реалізовувати інтерактивність і динамічні функції, забезпечуючи зручність і ефективність користування застосунком.

Даний розділ детально висвітлює особливості коду, який лежить в основі вебзастосунку, пояснює використані технології та принципи їхнього застосування. У процесі розробки було приділено значну увагу організації коду, дотриманню стандартів веброзробки та використанню сучасних підходів до програмування. Крім того, буде проаналізовано ключові сценарії взаємодії користувача з вебзастосунком і те, як вони реалізовані у вигляді програмного коду. Цей аналіз дозволить зрозуміти, як саме забезпечується динамічність і функціональність системи, а також оцінити ефективність використаних рішень.

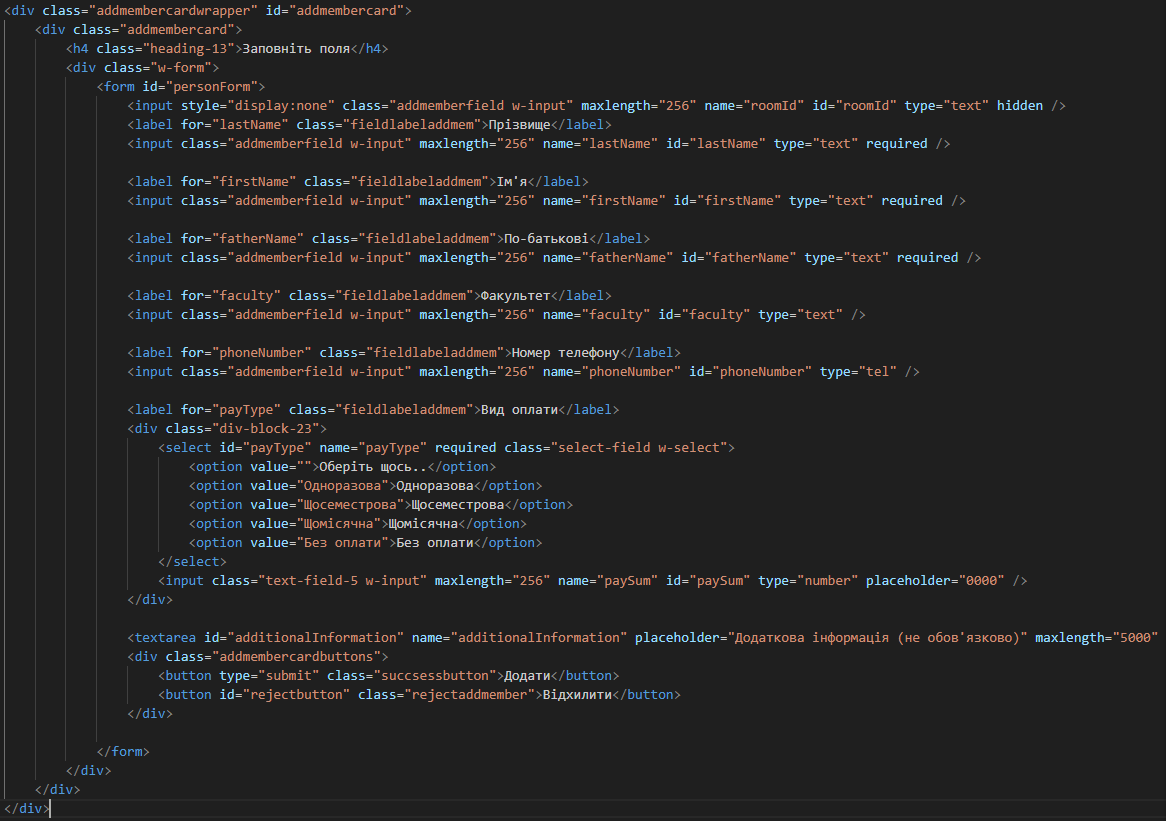
3.3.1 Огляд HTML коду

У процесі створення вебзастосунку використовувалась платформа Webflow, яка дозволила автоматично згенерувати значну частину HTML-коду. Це забезпечило швидкість і зручність розробки, але потребує детального аналізу для оцінки структури, відповідності стандартам і можливостей подальшого вдосконалення.

Згенерований код Webflow характеризується суворою структурованістю та чітким розподілом елементів. Для кожного компонента сторінки автоматично створюються контейнери, секції та блоки з унікальними класами, для асоціації з CSS. Семантичність HTML коду частково залежить від дій користувача при створенні структури, і в даному випадку помітно, що більшість елементів використовуються відповідно до їх призначення (наприклад, заголовки, списки, текстові блоки). Однак певні аспекти, такі як наявність зайвих div-елементів, потребують подальшого доопрацювання для зменшення складності DOM і оптимізації швидкості завантаження сторінки.

Код також включає спеціально створені класи для стилізації елементів, що є характерним для Webflow. Використання унікальних класів для кожного елемента може полегшити внесення змін через інтерфейс платформи, однак призводить до надлишковості в коді. Наприклад, замість повторного використання одного класу для однакових елементів генерується новий клас для кожної модифікації. Це може ускладнити ручну підтримку та редагування коду, якщо у майбутньому виникне потреба у значній адаптації чи перенесенні проекту до іншої платформи.

Позитивним аспектом згенерованого коду є його адаптивність. Завдяки автоматично інтегрованим стилям і грід-системі, сторінка виглядає коректно на різних пристроях і розмірах екранів. Також у коді застосовано атрибути для інтеграції з JavaScript, що дозволяє забезпечити динамічну взаємодію з користувачем. Однак, через автоматичний підхід можуть виникати складності із забезпеченням оптимальної продуктивності, що потребує ручного доопрацювання, зокрема, видалення непотрібних елементів або оптимізації скриптів.

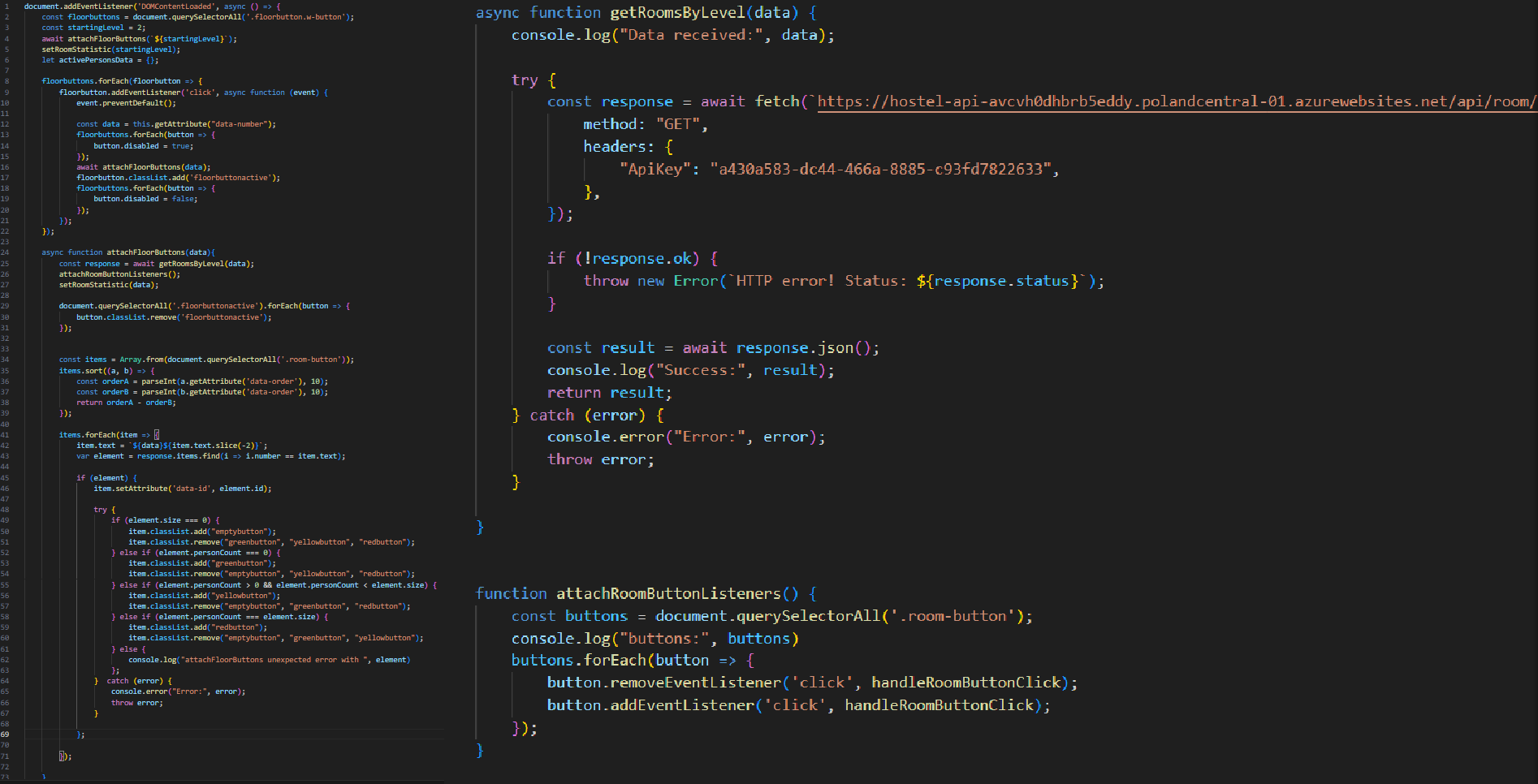
Рисунок 10. Приклад коду. Елемент addmembercard

Загалом, згенерований Webflow код має низку переваг, включаючи швидкість розробки, інтеграцію адаптивності та зручність стилізації. Водночас, для забезпечення найвищих стандартів продуктивності та відповідності кращим практикам розробки, він потребує оптимізації, зокрема, шляхом спрощення структури, зменшення надлишковості класів та перевірки семантичності елементів.

3.3.2 Опис JavaScript коду

Вибір поверху та відображення кімнат

Перша частина коду відповідає за реалізацію функціоналу вибору поверху, завантаження даних про кімнати з сервера, їх обробку та оновлення інтерфейсу. Основна мета коду — забезпечити динамічну інтерактивність для користувача під час взаємодії з кнопками поверхів і візуальне представлення стану кімнат.

Рисунок 11. Функції JS для відображення кімнат

У представленому вище коді описується функціонал взаємодії з елементами вебзастосунку для керування гуртожитком, зокрема динамічне оновлення кнопок поверхів, відображення кімнат та їхніх характеристик. Код структурований та складається з кількох функцій, кожна з яких виконує конкретну задачу. Нижче наведено розгорнутий аналіз функцій та їхньої ролі у вебзастосунку.

Загалом, виконання коду починається з реєстрації обробника події DOMContentLoaded, який спрацьовує після повного завантаження HTML-документу. Це забезпечує виконання основного скрипта лише після готовності DOM. У цьому обробнику відбувається ініціалізація взаємодії з елементами сторінки, такими як кнопки поверхів, та завантаження початкових даних. За допомогою методу querySelectorAll обираються всі кнопки поверхів із класом .floorbutton.w-button, що зберігаються у змінній floorbuttons. Для кожної кнопки встановлюється обробник події click, що дозволяє виконувати динамічне оновлення вмісту сторінки при натисканні.

Функція DOMContentLoaded на мові JS:

document.addEventListener('DOMContentLoaded', async () => {

    const floorbuttons = document.querySelectorAll('.floorbutton.w-button');

    const startingLevel = 2;

    await attachFloorButtons(`${startingLevel}`);

    setRoomStatistic(startingLevel);

    let activePersonsData = {};

    floorbuttons.forEach(floorbutton => {

        floorbutton.addEventListener('click', async function (event) {

            event.preventDefault();

            const data = this.getAttribute("data-number");

            floorbuttons.forEach(button => {

                button.disabled = true;

            });

            await attachFloorButtons(data);

            floorbutton.classList.add('floorbuttonactive');

            floorbuttons.forEach(button => {

                button.disabled = false;

            });

        });

    });

Наступна функція attachFloorButtons відповідає за динамічне завантаження та оновлення інформації про кімнати на обраному поверсі. Вона виконує свої дії покроково, забезпечуючи взаємозв'язок між даними з сервера та інтерфейсом користувача.

На початку функція викликає асинхронну функцію getRoomsByLevel, передаючи їй параметр data, що відповідає номеру поверху. Ця функція надсилає запит до сервера та повертає об'єкт із даними про кімнати на обраному поверсі. Після отримання відповіді викликається функція attachRoomButtonListeners, яка прив’язує події до кнопок, що представляють кімнати, забезпечуючи їхню інтерактивність. Далі функція setRoomStatistic оновлює статистичну інформацію для обраного поверху.

Після цього функція видаляє активний стан із усіх кнопок поверхів. Для цього вона знаходить усі елементи з класом floorbuttonactive і виконує метод classList.remove, щоб очистити цей клас.

Далі функція обробляє елементи, які представляють кімнати. Вона створює масив елементів, отриманих через селектор .room-button, і сортує їх за атрибутом data-order у порядку зростання. Це забезпечує правильну послідовність відображення кімнат у відповідності до їхньої нумерації.

Після сортування кожен елемент обробляється окремо. Для кожної кнопки кімнати значення її тексту змінюється відповідно до номера поверху, який додається до останніх двох символів наявного тексту. Потім функція намагається знайти відповідний об'єкт кімнати в отриманих із сервера даних, використовуючи метод find. Якщо відповідний об'єкт знайдено, до елемента додається атрибут data-id, який відповідає унікальному ідентифікатору кімнати.

Далі виконується перевірка стану кімнати за допомогою конструкції try...catch, яка забезпечує обробку можливих помилок під час виконання. В залежності від розміру кімнати (size) та кількості мешканців (personCount), до елемента застосовуються різні класи для позначення стану кімнати. Якщо кімната порожня (розмір дорівнює 0), до елемента додається клас emptybutton, а інші класи видаляються. Якщо кімната має вільні місця, застосовується клас greenbutton. Якщо кімната частково зайнята, використовується клас yellowbutton, а якщо повністю заповнена — клас redbutton. У разі виникнення помилки функція виводить її у консоль і припиняє виконання через повторне генерацію помилки за допомогою throw.

У підсумку функція attachFloorButtons забезпечує інтеграцію серверних даних із інтерфейсом користувача, створюючи динамічну та адаптивну систему для управління поверхами та кімнатами.

У функції attachFloorButtons виклики інших функцій реалізовано за допомогою композиції функцій, що є одним із ключових принципів функціонального програмування. Цей метод дозволяє розділити складну задачу на менші підзадачі, кожна з яких виконує конкретну функцію, а їх взаємодія забезпечує загальний результат. У даному випадку кожна викликана функція відповідає за певний аспект роботи, а їх послідовне виконання формує завершений функціонал.

Напочатку функції використовується виклик асинхронної функції getRoomsByLevel(data) для отримання даних із сервера. Цей метод є прикладом чистої функції, яка виконує конкретну дію (отримання даних) і повертає результат без побічних ефектів. Результат цієї функції (response) стає вхідними даними для наступних дій.

Далі викликаються функції attachRoomButtonListeners та setRoomStatistic, які забезпечують оновлення функціональності кнопок кімнат і встановлення відповідної статистики. Ці функції виконуються безпосередньо, оскільки їх результат не впливає на наступні етапи роботи, але вони сприяють забезпеченню інтерактивності інтерфейсу.

Усередині циклу для кожного елемента кімнати функція find використовується для пошуку відповідного об'єкта в даних, отриманих із сервера. Це є прикладом високого порядку функції, оскільки find приймає функцію як параметр і виконує її для кожного елемента масиву.

Таким чином, у функції attachFloorButtons послідовно використовуються чисті функції, функції високого порядку та модульний підхід до розподілу завдань. Це забезпечує структурованість, зручність підтримки та зрозумілість коду, що є важливим для масштабованих програмних систем.

Функція attachFloorButtons є асинхронною і призначена для обробки даних, отриманих з API. Ця функція приймає параметр data, який визначає номер поверху.

Функція attachFloorButtons на мові JS:

async function attachFloorButtons(data){

        const response = await getRoomsByLevel(data);

        attachRoomButtonListeners();

        setRoomStatistic(data);

        document.querySelectorAll('.floorbuttonactive').forEach(button => {

            button.classList.remove('floorbuttonactive');

        });

        const items = Array.from(document.querySelectorAll('.room-button'));

        items.sort((a, b) => {

            const orderA = parseInt(a.getAttribute('data-order'), 10);

            const orderB = parseInt(b.getAttribute('data-order'), 10);

            return orderA - orderB;

        });

        items.forEach(item => {

            item.text = `${data}${item.text.slice(-2)}`;

            var element = response.items.find(i => i.number == item.text);

            if (element) {

                item.setAttribute('data-id', element.id);

                try {

                    if (element.size === 0) {

                        item.classList.add("emptybutton");

                        item.classList.remove("greenbutton", "yellowbutton", "redbutton");

                    } else if (element.personCount === 0) {

                        item.classList.add("greenbutton");

                        item.classList.remove("emptybutton", "yellowbutton", "redbutton");

                    } else if (element.personCount > 0 && element.personCount < element.size) {

                        item.classList.add("yellowbutton");

                        item.classList.remove("emptybutton", "greenbutton", "redbutton");

                    } else if (element.personCount === element.size) {

                        item.classList.add("redbutton");

                        item.classList.remove("emptybutton", "greenbutton", "yellowbutton");

                    } else {

                        console.log("attachFloorButtons unexpected error with ", element)

                    };

                }  catch (error) {

                    console.error("Error:", error);

                    throw error;

                }

            };

});

    }

Функція getRoomsByLevel, яка викликається всередині attachFloorButtons здійснює запит до серверного API для отримання інформації про кімнати на вказаному поверсі. Після отримання відповіді функція виконує дії з елементами кнопок кімнат: сортує їх за атрибутом data-order, динамічно змінює їхній текст та додає стилі відповідно до заповненості кімнат. Для цього використовуються класи emptybutton, greenbutton, yellowbutton, redbutton, що вказують на статус кімнати (порожня, частково заповнена чи повністю зайнята).

Напочатку функція виводить до консолі параметр data, який отримано як вхідний аргумент. Це забезпечує відстеження, які дані передаються для формування запиту.

Далі функція здійснює HTTP-запит методом GET за допомогою функції fetch. URL-адреса формується динамічно, включаючи параметр data, який відповідає номеру поверху, а також статичний ідентифікатор гуртожитку (hostelId) і API-ключ для автентифікації. В заголовках запиту передається API-ключ, що забезпечує доступ до необхідних ресурсів на сервері.

Після виконання запиту перевіряється статус відповіді. Якщо відповідь сервера не є успішною (статус код не входить у діапазон 200–299), генерується помилка за допомогою оператора throw. Це дозволяє обробляти можливі помилки, наприклад, недоступність сервера або неправильну URL-адресу.

Якщо запит виконано успішно, відповідь обробляється методом json() для перетворення отриманих даних у формат JSON. Цей процес є асинхронним, тому функція очікує завершення цієї операції за допомогою оператора await. Отримані дані зберігаються в змінній result.

Дані про успішну відповідь також виводяться до консолі для налагодження, після чого результат повертається як вихідне значення функції.

У разі виникнення помилки під час виконання запиту або обробки даних вона обробляється в блоці catch. У цьому випадку повідомлення про помилку виводиться до консолі, а помилка повторно генерується оператором throw, що дозволяє обробляти її на рівні виклику функції.

Функція getRoomsByLevel на мові JS:

    async function getRoomsByLevel(data) {

        console.log("Data received:", data);

        try {

            const response = await fetch(`https://hostel-api-avcvh0dhbrb5eddy.polandcentral-01.azurewebsites.net/api/room/items?hostelId=B05862FB-3AB5-433A-B6EE-08DD07B46EC6&level=${data}`, {

                method: "GET",

                headers: {

                    "ApiKey": "a430a583-dc44-466a-8885-c93fd7822633",

                },

            });

            if (!response.ok) {

                throw new Error(`HTTP error! Status: ${response.status}`);

            }

            const result = await response.json();

            console.log("Success:", result);

            return result;

        } catch (error) {

            console.error("Error:", error);

            throw error;

        }

    }

Окрім основних функцій, у коді визначено допоміжну функцію attachRoomButtonListeners, яка додає обробники подій для кнопок кімнат. Ці обробники забезпечують динамічну взаємодію користувача з елементами інтерфейсу.

На початку функція за допомогою document.querySelectorAll знаходить усі елементи сторінки, які мають клас room-button. Ці елементи зберігаються у змінній buttons, що містить список вузлів DOM. Для налагодження отриманий список виводиться в консоль, що дозволяє перевірити правильність вибору елементів.

Далі для кожної кнопки з цього списку виконується операція через метод forEach. В рамках цієї операції функція спочатку видаляє будь-який існуючий обробник подій click, пов’язаний із функцією handleRoomButtonClick, використовуючи метод removeEventListener. Це необхідно для запобігання дублюванню обробників у разі багаторазового виклику attachRoomButtonListeners, що може призвести до некоректної роботи інтерфейсу (наприклад, багаторазової обробки одного кліку).

Після видалення попередніх обробників до кожної кнопки додається новий обробник подій click за допомогою методу addEventListener. Цей обробник посилається на функцію handleRoomButtonClick, яка відкриває картки з інформацією про кімнату.

Функція attachRoomButtonListeners на мові JS:

    function attachRoomButtonListeners() {

        const buttons = document.querySelectorAll('.room-button');

        console.log("buttons:", buttons)

        buttons.forEach(button => {

            button.removeEventListener('click', handleRoomButtonClick);

            button.addEventListener('click', handleRoomButtonClick);

        });

    }

Також до функцій, які викликаються у attachFloorButtons є функції для показу статистики поселення в гуртожитку. Перша функція getRoomStatistic є асинхронною і відповідає за отримання статистичних даних щодо кімнат на певному поверсі. Вона здійснює HTTP-запит до API сервера, використовуючи метод fetch, з параметрами, що включають рівень (номер поверху) та ідентифікатор гуртожитку. Після того, як дані отримано, функція перевіряє, чи відповідає статус запиту успішному результату (код статусу HTTP 200). Якщо відповідь неуспішна, вона намагається обробити повідомлення про помилку, яке приходить від сервера, та повертає об’єкт з полем success: false, що включає повідомлення про помилку та код помилки. Якщо статус запиту успішний, відповідь конвертується в формат JSON, і виводяться отримані дані про статистику кімнат, зокрема кількість мешканців на рівні та в гуртожитку. Функція повертає об’єкт з полем success: true і самими даними про статистику.

Друга функція setRoomStatistic відповідає за відображення статистичних даних на інтерфейсі після того, як вони були отримані через функцію getRoomStatistic. Спочатку вона викликає getRoomStatistic і чекає на результат, використовуючи метод .then(), щоб обробити отриману відповідь. Якщо результат успішний, функція оновлює кілька елементів на сторінці за допомогою значень з отриманих даних: кількість мешканців на поточному поверсі (personCountByLevel) і в гуртожитку загалом (personCountByHostel). Окрім цього, функція обчислює відсоткове заповнення поверху та гуртожитку за допомогою значень personPercentageLevelByLevel та personPercentageLevelByHostel, і оновлює прогрес-бар на сторінці відповідно до цих значень. У разі невдачі запиту на отримання статистики, в консоль виводиться помилка з повідомленням, отриманим від сервера.

Функції getRoomStatistic та setRoomStatistic на мові JS:

async function getRoomStatistic(level) {

        try {

            const response = await fetch(`https://hostel-api-avcvh0dhbrb5eddy.polandcentral-01.azurewebsites.net/api/room/statistic?level=${level}&hostelId=B05862FB-3AB5-433A-B6EE-08DD07B46EC6`, {

                method: 'GET',

                headers: {

                   "ApiKey": "a430a583-dc44-466a-8885-c93fd7822633",

                },

            });

            if (!response.ok) {

                const errorResponse = await response.json();

                console.error('Error:', errorResponse.message);

                return {

                    success: false,

                    message: errorResponse.message,

                    errorCode: errorResponse.errorCode,

                };

            }

            const data = await response.json();

            console.log('Room Statistic:', data.item);

            return {

                success: true,

                data: data.item,

            };

        } catch (error) {

            console.error('Fetch Room Statistic error:', error);

            return {

                success: false,

                message: error.message,

            };

        }

    }

    function setRoomStatistic(level){

        getRoomStatistic(level).then((result) => {

            if (result.success) {

                document.getElementById('floorMembersCount').textContent = result.data.personCountByLevel;

                document.getElementById('allMembersCount').textContent = result.data.personCountByHostel;

                const percentageLevel = Math.round(result.data.personPercentageLevelByLevel);

                const percentageHostel = Math.round(result.data.personPercentageLevelByHostel);

                document.getElementById('percentageLevel').textContent = `${percentageLevel}%`;

                document.getElementById('percentageAll').textContent = `${percentageHostel}%`;

                document.getElementById('progressBarFloor').style.width = `${percentageLevel}%`;

                document.getElementById('progressBarDorm').style.width = `${percentageHostel}%`;

            } else {

                console.error('Помилка отримання статистики:', result.message);

            }

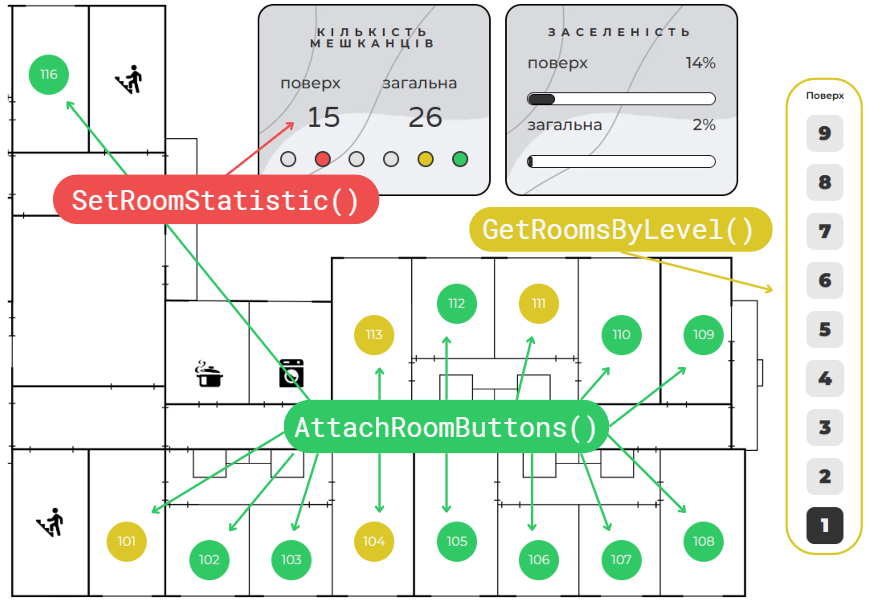
    });

Рисунок 12. Результати роботи функцій

Таким чином, ці функції разом дозволяють динамічно отримувати і відображати статистичні дані про заповненість кімнат на конкретному поверсі та в гуртожитку в цілому, що є важливою частиною інтерфейсу для користувачів і адміністрації.

Відображення сайдбару з інформацією про кімнату

Відображення сайдбару з інформацією про вибрану кімнату реалізується через взаємодію користувача з кнопкою кімнати та виклик відповідного функціоналу. Головним елементом цього процесу є функція onDisplayBlock, яка відповідає за ініціалізацію відображення потрібного сайдбару, підготовку його до роботи, завантаження інформації про кімнату та інтеграцію цієї інформації в інтерфейс.

Рисунок 13. Картка кімнати з мешканцем

Ця функція виконує основну роботу з відкриттям картки кімнати. Спочатку викликається функція closeRMCDescription, яка приховує всі існуючі картки опису кімнат (rmcdescription) у вигляді сайдбару. Ця функція знаходить усі елементи з класом rmcdescription і приховує їх, змінюючи стиль display на "none". Це необхідно, щоб перед відкриттям нової картки кімнати закрити вже відкриті картки мешканців, якщо вони є.

Функція closeRMCDescription на мові JS:

    function closeRMCDescription(){

        const cards = document.querySelectorAll('.rmcdescription');

        cards.forEach(item => {

            item.style.display = "none";

        })

    }

Наступною викликається функція fetchRoomDataAndUpdateUI, яка відповідає за отримання даних кімнати з сервера та оновлення інтерфейсу на основі цих даних.

Функція fetchRoomDataAndUpdateUI відповідає за отримання інформації про вибрану кімнату з сервера та оновлення інтерфейсу користувача відповідно до отриманих даних. Вона виконує асинхронний HTTP-запит до API та обробляє отриману відповідь для інтеграції даних у сайдбар кімнати.

На першому етапі функція ініціює HTTP-запит методом GET до вказаного API за допомогою функції fetch. У запиті передається ідентифікатор кімнати через параметр id, а також у заголовку додається API-ключ для авторизації. Цей запит спрямований на отримання детальної інформації про кімнату.

Далі функція перевіряє статус відповіді від сервера. Якщо відповідь є неуспішною (тобто статус код не вказує на успіх), викликається виключення із повідомленням про помилку. Це дозволяє обробляти такі ситуації, як відсутність доступу до API чи помилки на стороні сервера.

У випадку успішного запиту функція розбирає відповідь у форматі JSON, отримуючи необхідні дані. Отриманий об'єкт результату передається до функції updateRoomCardContent, яка відповідає за оновлення інтерфейсу користувача. У цьому випадку дані кімнати відображаються в сайдбарі, забезпечуючи інтерактивну роботу користувача із системою.

Якщо виникає будь-яка помилка під час виконання запиту або обробки даних, вона обробляється у блоці catch. Помилка реєструється в консолі для подальшого аналізу розробниками.

Таким чином, функція fetchRoomDataAndUpdateUI забезпечує динамічну інтеграцію актуальної інформації про кімнату в інтерфейс.

Функція fetchRoomDataAndUpdateUI на мові JS:

async function fetchRoomDataAndUpdateUI(data) {

        try {

            const response = await fetch(`https://hostel-api-avcvh0dhbrb5eddy.polandcentral-01.azurewebsites.net/api/room/item?id=${data}`, {

                method: "GET",

                headers: {

                    "ApiKey": "a430a583-dc44-466a-8885-c93fd7822633",

                },

            });

            if (!response.ok) {

                throw new Error('Помилка при отриманні даних');

            }

            const result = await response.json();

            console.log('Отримані дані:', result);

            updateRoomCardContent(result.item);

        } catch (error) {

            console.error('Помилка:', error);

        }

    }

**Після того, як дані були отримані, відбувається базове оновлення основного блоку сайдбару кімнати за допомогою функції updateRoomCardContent, вона отримує як аргумент об'єкт data, що містить інформацію про кімнату. Спочатку функція оновлює номер кімнати, задаючи текстовий вміст елемента з класом room-number у вигляді "Кімната №" та номер кімнати, що міститься у властивості data.number. Потім оновлюється додаткова інформація про кімнату. Значення поля data.additionalInformation вставляється у відповідний елемент із ідентифікатором roomcardinfo.**

**Функція updateRoomCardContent на мові JS:**

    function updateRoomCardContent(data) {

        document.querySelector('.room-number').textContent = `Кімната №${data.number}`;

        document.querySelector('#roomcardinfo').textContent = `${data.additionalInformation}`;

        updateRoomMembers(data);

    }

**Нарешті, викликається функція updateRoomMembers для детального оновлення інформації про мешканців кімнати. Ця функція відповідає за виведення списку мешканців кімнати у відповідному контейнері. Спочатку отримується контейнер для відображення мешканців кімнати за допомогою ідентифікатора room-members. Також вибираються кнопки взаємодії з мешканцями через клас roomcardbuttons. Дані про мешканців кімнати зберігаються у змінну activePersonsData для подальшого використання. Далі виконується ітерація через кількість місць у кімнаті (змінна data.size). Для кожного місця перевіряється наявність мешканця у відповідному місці (дані беруться з масиву data.persons). Якщо мешканець є, інформація про нього (ім'я, факультет, номер телефону, тип оплати тощо) відображається у відповідних елементах.**

**Зокрема:**

1. **ПІБ мешканця відображається у блоці з класом paragraph.**
2. **Дані про факультет, номер телефону та оплату оновлюються у відповідних елементах за їхніми ідентифікаторами.**
3. **Додаткова інформація перевіряється на наявність та відображається лише у випадку, якщо вона задана.**

**Місце, зайняте мешканцем, стає видимим, а маркер "вільне місце" приховується. Для кнопок взаємодії додається атрибут member-id, що відповідає унікальному ідентифікатору мешканця. Якщо місце порожнє, маркер "вільне місце" стає видимим, а блок із інформацією про мешканця приховується.**

**Функція updateRoomMembers на мові JS:**

**function updateRoomMembers(data) {**

        const roomContainer = document.getElementById('room-members');

        const roomCardButtons = document.querySelectorAll('.roomcardbuttons');

        console.log('activePd -----', data);

        activePersonsData = data.persons;

        for (let i = 0; i < data.size; i++) {

            const memberData = data.persons[i];

            console.log("memberData", memberData);

            if (memberData) {

                const roomMember = roomContainer.querySelector(`#room-member-${i}`);

                roomMember.querySelector('.paragraph').textContent = `${memberData.lastName} ${memberData.firstName} ${memberData.fatherName}`;

                document.getElementById(`faculty-${i}`).textContent = `${memberData.faculty}`;

                document.getElementById(`phoneNumber-${i}`).textContent = `${memberData.phoneNumber}`;

                document.getElementById(`payement-${i}`).textContent = `${memberData.payType} - ${memberData.paySum}`;

                if (memberData.additionalInformation && memberData.additionalInformation.trim() !== '-') {

                    document.getElementById(`additionalInfo-${i}`).style.display = "block";

                    document.getElementById(`additionalInfo-${i}`).textContent = `${memberData.additionalInformation}`;

                }else{

                    document.getElementById(`additionalInfo-${i}`).style.display = "none";

                }

                roomMember.style.display = "block";

                document.getElementById(`free-space-${i}`).style.display = "none";

                roomCardButtons[i].setAttribute('member-id', memberData.id);

            } else {

                document.getElementById(`room-member-${i}`).style.display = "none";

                document.getElementById(`free-space-${i}`).style.display = "block";

            }

        }

    }

**Html елементи, з якими вона взаємодіє:**

**<div id="room-members">**

<div id="room-member-0" class="room-memeber-card w-clearfix" style="display:none">

<p class="paragraph">ПІБ</p>

      <div data-member-id="0" style=" user-select: none; display: block;"

class="open-member-info">

<div class="text-block-8">&gt;</div>

</div>

      </div>

      <div id="description-0" style="display:none" class="room-memeber-card rmcdescription">

             <div class="firstrowofdescrioption">

              <div class="div-block-20 divleft">

                    <div class="text-block-5" id="faculty-0">фак</div>

                   </div>

                   <div class="div-block-20 divleft">

                         <div class="text-block-4" id="phoneNumber-0">номер.тел</div>

                   </div>

                <div class="div-block-20">

                    <div class="text-block-6" id="payement-0">оплата</div>

                  </div>

</div>

            <div class="secondrowofdecription">

            <div id="additionalInfo-0">Якась додаткова інформація</div>

            </div>

            <div class="roomcardbuttons">

                  <div class="succsessbutton suc1" style="user-select: none">

                        <div>Переселити</div>

                   </div>

                   <div class="editbutton" style="user-select: none">

                      <div class="text-block-9">Редагувати</div>

                  </div>

                  <div class="deletebutton" style="user-select: none">

                        <div class="text-block-9">Виселити</div>

                  </div>

            </div>

 </div>

**<div class="room-memeber-card w-clearfix emptycard" id="free-space-0" style="user-select: none">**

<p class="paragraph">Вільне місце</p>

</div>

**Наступним кроком у кожному елементі мешканця (roomcardbuttons) додаються обробники подій для кнопок "Переселити", "Редагувати" і "Виселити". Ці кнопки викликають функцію** handleMemberButtonClick **з відповідною дією (move, edit, delete), передаючи ідентифікатори мешканця та кімнати. Додатково, для кнопок додавання нового мешканця (button-add-member) виконується прив'язка до ідентифікатора кімнати. При натисканні на ці кнопки викликається асинхронна функція** handleAddMemberButtonClick**, яка відповідає за обробку логіки додавання нового мешканця до кімнати.**

**Таким чином, ці функції спільно забезпечують повне оновлення інформації про кімнату та її мешканців, динамічно адаптуючи інтерфейс відповідно до поточного стану даних.**

**Функція onDisplayBlock на мові JS:**

**function onDisplayBlock(roomId) {**

        console.log(`Отримуємо дані для кімнати з ID: ${roomId}`);

        closeRMCDescription();

        fetchRoomDataAndUpdateUI(roomId); // Викликаємо функцію для отримання даних

        document.querySelectorAll('.roomcardbuttons').forEach(card => {

            ;

            card.querySelector('.suc1')?.addEventListener('click', () => {

                handleMemberButtonClick('move', card.getAttribute('member-id'), roomId);

            });

            card.querySelector('.editbutton')?.addEventListener('click', () => {

                handleMemberButtonClick('edit', card.getAttribute('member-id'), roomId);

            });

            card.querySelector('.deletebutton')?.addEventListener('click', () => {

                handleMemberButtonClick('delete', card.getAttribute('member-id'), roomId);

            });

        const addMemberButtons = document.querySelectorAll('.button-add-member'); // Тільки з атрибутом data-id

            addMemberButtons.forEach(addMemberButton => {

                addMemberButton.setAttribute('room-id', roomId)

                addMemberButton.addEventListener('click', async function (event) {

                    await handleAddMemberButtonClick(roomId);

                })

            })

        });

    }

Додавання мешканців

Функція handleAddMemberButtonClick відповідає за обробку події натискання кнопки додавання мешканця до кімнати. Вона виконується асинхронно та включає кілька ключових кроків для оновлення інтерфейсу та виконання серверних операцій.

На початку функція виводить у консоль ID кімнати, яке передається як аргумент roomId. Це допомагає відстежити, яку саме кімнату обрано для додавання нового мешканця. Далі, використовуючи метод document.querySelector('#roomId'), функція знаходить елемент на сторінці, який має ID roomId, і змінює його текстовий вміст (властивість textContent) на значення roomId. Це оновлює значення ID для елемента в HTML коді.

Функція handleAddMemberButtonClick на мові JS:

    async function handleAddMemberButtonClick(roomId) {

        console.log(`Room ID handle: ${roomId}`);

        console.log(document.querySelector('#roomId'))

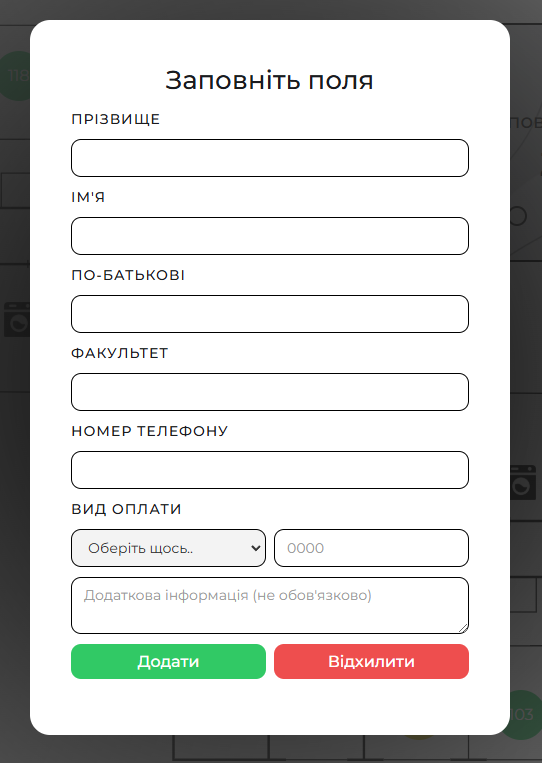
        document.querySelector('#roomId').textContent = roomId;

        await addMember(roomId);

        await hideAddMemberCard();

    }

Після цього викликається асинхронна функція addMember(roomId), яка відповідає за додавання нового мешканця до кімнати. Оскільки це асинхронна функція, виконується очікування її завершення за допомогою ключового слова await. Вона отримує ідентифікатор кімнати (roomId) як вхідний параметр і реалізує логіку взаємодії з формою введення даних про мешканця.

Рисунок 14. Форма для додавання мешканця

Спершу функція отримує доступ до елементів форми через її ідентифікатор personForm. Для кнопки скасування (rejectButton) додається обробник події click, який перешкоджає стандартній поведінці кнопки, скидає поля форми за допомогою методу reset() та приховує форму за допомогою функції hideAddMemberCard.

Далі додається обробник події submit для форми, який забороняє стандартну відправку форми і виконує обробку даних. Зібрані дані з полів форми, такі як прізвище, ім’я, по батькові, факультет, номер телефону, тип та сума оплати, а також додаткова інформація, формуються у вигляді об’єкта formData, до якого додається ідентифікатор кімнати.

Після цього здійснюється спроба відправки POST-запиту на сервер за вказаним URL для створення нового мешканця. У запиті вказуються метод POST, заголовки з інформацією про тип контенту та API-ключ, а також тіло запиту, представлене у форматі JSON.

У випадку успішного виконання запиту користувача інформують про додавання нового мешканця за допомогою повідомлення alert. У разі помилки користувач отримує відповідне повідомлення, а інформація про помилку записується до консолі.

Після завершення операції форма скидається, ховається за допомогою функції hideAddMemberCard, і викликається функція fetchRoomDataAndUpdateUI, яка оновлює дані про мешканців кімнати. Таким чином, функція забезпечує інтерактивне додавання мешканців до кімнати через форму з дотриманням принципів асинхронності та обробки можливих помилок.

Функція addMember на мові JS:

    async function addMember(roomId){

        const form = document.getElementById('personForm');

        const rejectButton = document.getElementById('rejectbutton');

        rejectButton.addEventListener('click', function (event) {

            event.preventDefault(); // Зупиняє стандартну поведінку кнопки

            form.reset(); // Скидає всі поля форми

            hideAddMemberCard();

        });

        form.addEventListener('submit', async (event) => {

            event.preventDefault();

            const formData = {

                person: {

                    lastName: document.getElementById('lastName').value,

                    firstName: document.getElementById('firstName').value,

                    fatherName: document.getElementById('fatherName').value,

                    faculty: document.getElementById('faculty').value,

                    phoneNumber: document.getElementById('phoneNumber').value,

                    payType: document.getElementById('payType').value,

                    paySum: document.getElementById('paySum').value,

                    additionalInformation: document.getElementById('additionalInformation').value,

                },

                roomId: roomId

            };

            console.log(formData);

            try {

                const response = await fetch('https://hostel-api-avcvh0dhbrb5eddy.polandcentral-01.azurewebsites.net/api/person/create', {

                    method: 'POST',

                    headers: {

                        'Content-Type': 'application/json',

                        "ApiKey": "a430a583-dc44-466a-8885-c93fd7822633",

                    },

                    body: JSON.stringify(formData),

                });

                if (response.ok) {

                    console.log('Форма успішно надіслана');

                    alert('Додано мешканця!');

                    return;

                } else {

                    console.error('Помилка під час надсилання форми');

                    alert('Помилка під час надсилання форми');

                    return;

                }

            } catch (error) {

                console.error('Виникла помилка:', error);

            }

            form.reset();

            hideAddMemberCard();

            fetchRoomDataAndUpdateUI(roomId);

        });

    }

Наступним кроком функція handleAddMemberButtonClick викликає ще одну асинхронну функцію — hideAddMemberCard, яка відповідає за приховування або закриття картки додавання мешканця в інтерфейсі. Після того, як ця функція виконується, користувач більше не бачить форму для додавання мешканця, що свідчить про завершення процесу. Реалізація цієї функції дуже проста, вона знаходить елемент в HTML коді сторінки, до якого прив’язана картка для поселення і приховує його.

Така реалізація забезпечує адаптивний підхід до зміни елементів сторінки на основі отриманих даних та дозволяє гнучко змінювати стан сторінки залежно від введених даних чи дій користувача.

Переселення, Редагування чи Виселення

Функція handleMemberButtonClick забезпечує обробку натискання кнопок дій, пов’язаних із редагуванням, переселенням або виселенням мешканця, що викликається в контексті функції onDisplayBlock. Основною метою функції є реалізація логіки виконання відповідної дії для мешканця, ідентифікованого за memberId, у контексті конкретної кімнати, ідентифікатор якої передається через roomId.

Функція отримує параметр action, який визначає, яку саме дію необхідно виконати. На основі значення цього параметра виконується одна з трьох опцій:

* 1. **Переселення (**move**)**:

Для реалізації цієї дії функція запитує номер нової кімнати через функцію requestRoomNumber. Після цього, за допомогою функції findRoomByNumber, визначається ідентифікатор кімнати, до якої потрібно переселити мешканця. На основі отриманих даних створюється об'єкт formData, який містить ідентифікатор мешканця (PersonId) та ідентифікатор нової кімнати (roomId).

* 1. **Редагування (**edit**)**:

У випадку вибору дії редагування функція викликає editMemberInformation, яка дозволяє оновити інформацію про мешканця. Після цього формується об'єкт formData з оновленими даними для подальшого їх збереження.

* 1. **Виселення (**delete**)**:

Перед виконанням виселення функція показує діалогове вікно підтвердження через confirm. Якщо користувач скасовує операцію, виконання функції завершується. У разі підтвердження формується об'єкт formData, де вказується ідентифікатор мешканця (PersonId), а поле RoomId отримує значення null, що означає видалення мешканця з кімнати.

Після формування об'єкта formData виконується запит до сервера через метод fetch. Для запиту використовується метод POST, заголовки містять тип контенту (application/json) і API-ключ. У разі успішного виконання запиту виводиться повідомлення до консолі, а також викликається функція closeRMCDescription, що закриває опис мешканця, та функція fetchRoomDataAndUpdateUI, яка оновлює дані кімнати в інтерфейсі. У разі помилки виводиться відповідне повідомлення в консоль. Якщо під час виконання запиту виникає помилка, вона перехоплюється у блоці catch і записується до консолі.

Функція забезпечує централізоване керування діями над мешканцем у межах вебзастосунку, та уніфікує процес редагування даних мешканця, переселення до іншої кімнати або повне виселення.

Функція handleMemberButtonClick на мові JS:

    async function handleMemberButtonClick(action, memberId, roomId) {

        let formData = {};

        switch (action) {

            case 'move':

                console.log(`Перемістити member-id: ${memberId}`);

                let roomNumber = await requestRoomNumber();

                let roomIdF = await findRoomByNumber(roomNumber);

                console.log('id in handleMBC', roomIdF);

                formData = {

                    PersonId: memberId,

                    roomId: roomIdF

                };

                break;

            case 'edit':

                console.log(`Редагувати для member-id: ${memberId}`);

                formData = await editMemberInformation(memberId, roomId);

                break;

            case 'delete':

                const confirmDelete = confirm("Ви впевнені, що хочете виселити цього мешканця?");

                if (!confirmDelete) {

                     return; // Якщо користувач скасував, нічого не робимо

                };

                console.log(`Видалити для member-id: ${memberId}`);

                formData = {

                    PersonId: memberId,

                    RoomId: null

                };

                break;

            default:

                console.error(`Невідома дія: ${action}`);

        }

        console.log("надсилання форми", formData);

        try {

            const response = await fetch('https://hostel-api-avcvh0dhbrb5eddy.polandcentral-01.azurewebsites.net/api/person/save', {

                method: 'POST',

                headers: {

                    'Content-Type': 'application/json',

                    "ApiKey": "a430a583-dc44-466a-8885-c93fd7822633",

                },

                body: JSON.stringify(formData),

            });

            if (response.ok) {

                console.log('Форма успішно надіслана');

                closeRMCDescription();

                fetchRoomDataAndUpdateUI(roomId);

            } else {

                console.error('Помилка під час надсилання форми');

            }

        } catch (error) {

            console.error('Виникла помилка:', error);

        }

    }

Функції requestRoomNumber та findRoomByNumber забезпечують логіку отримання номера кімнати від користувача та пошук її ідентифікатора в базі даних. Вони є важливими етапами процесу переселення мешканця.

**Функція** requestRoomNumber викликає спливаюче вікно для введення номера кімнати. Якщо користувач вводить значення, яке не є порожнім і не було скасоване, воно перетворюється на ціле число (parseInt) і повертається для подальшого використання. Додатково функція виводить повідомлення з введеним номером в консоль і сповіщає користувача через діалогове вікно alert. Якщо ж номер не введено або операцію скасовано, функція виводить відповідне повідомлення в консоль і інформує користувача про відсутність введення.

Функція requestRoomNumber на мові JS:

    function requestRoomNumber() {

        const roomNumber = prompt("Введіть номер кімнати:", ""); // Спливаюче вікно із запитом

        if (roomNumber !== null && roomNumber.trim() !== "") {

            console.log(`Ви ввели номер кімнати: ${roomNumber}`);

            alert(`Ви ввели номер кімнати: ${roomNumber}`);

            return parseInt(roomNumber, 10);

            // Тут можна додати логіку, що робити з отриманим номером

        } else {

            console.log("Номер кімнати не введено або скасовано.");

            alert("Ви не ввели номер кімнати.");

        }

    }

Функція findRoomByNumber: Ця асинхронна функція виконує запит на сервер для отримання списку кімнат у визначеному гуртожитку. Вона надсилає запит до API за методом GET і використовує API-ключ для авторизації. Отримавши відповідь від сервера, функція перевіряє її на успішність. У разі успішного виконання результат запиту обробляється для пошуку кімнати за номером, переданим як аргумент. Якщо кімната знайдена, її ідентифікатор повертається для подальшого використання. У разі виникнення помилки під час виконання запиту функція обробляє її в блоці catch, записуючи відповідну інформацію до консолі.

Функція findRoomByNumber на мові JS:

    async function findRoomByNumber(roomNumber) {

        try {

            const response = await fetch(`https://hostel-api-avcvh0dhbrb5eddy.polandcentral-01.azurewebsites.net/api/room/items?hostelId=B05862FB-3AB5-433A-B6EE-08DD07B46EC6`, {

                method: "GET",

                headers: {

                    "ApiKey": "a430a583-dc44-466a-8885-c93fd7822633",

                },

            });

            if (!response.ok) {

                throw new Error('Помилка при отриманні даних');

            }

            const result = await response.json();

            room = result.items.find(item => item.number === roomNumber);

            console.log('знайдено кімнату:', room);

            return room.id;

        } catch (error) {

            console.error('Помилка:', error);

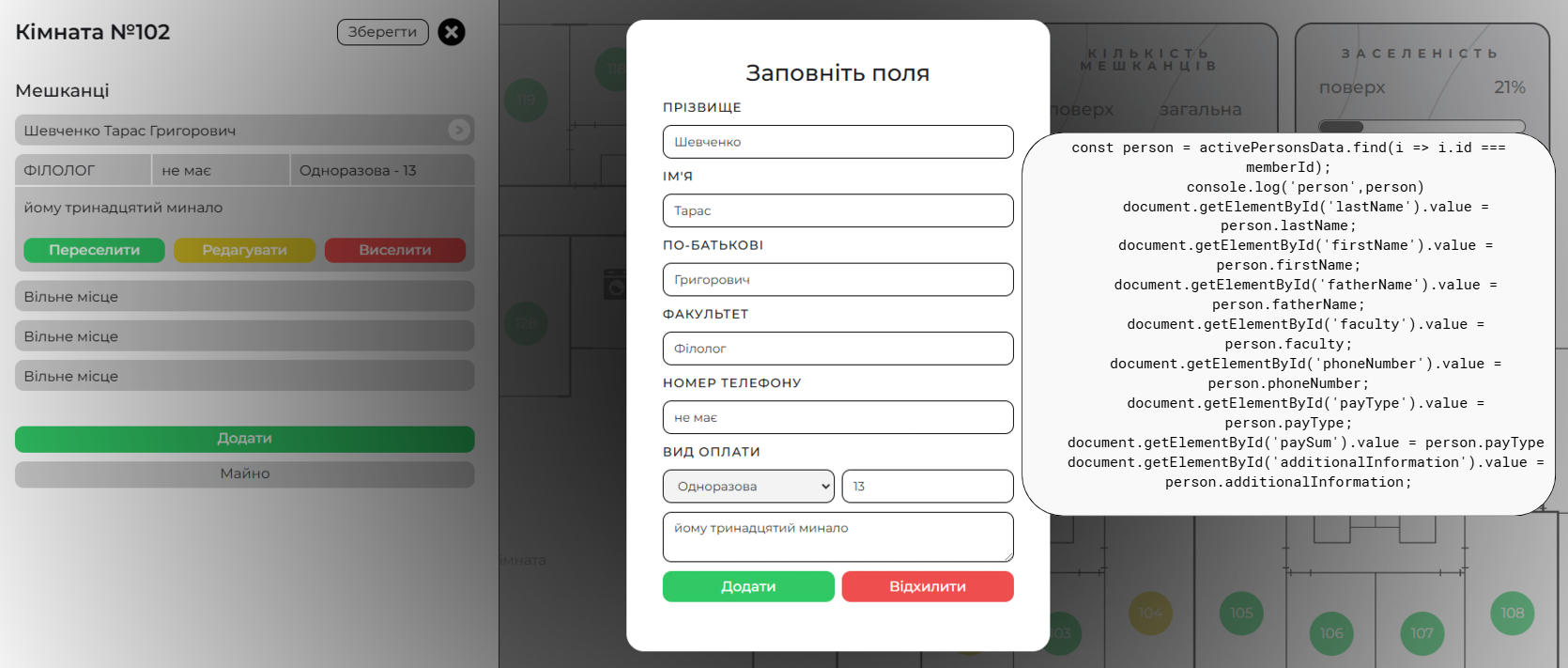
        }

    }

Ці дві функції забезпечують взаємодію користувача із системою для отримання номера кімнати та пошук її відповідності в базі даних, що є необхідним кроком для переселення мешканця.

Функція editMemberInformation забезпечує редагування інформації про мешканця, дозволяючи користувачеві оновити його дані через форму. Вона виконує кілька основних дій, які спрямовані на відображення поточних даних про мешканця, обробку змін і підготовку оновлених даних для подальшого використання.

Спочатку викликається функція openAddMemberCard, яка відкриває форму для введення інформації про мешканця. Це забезпечує доступ користувача до полів, які потрібно змінити. Для кнопки скасування (rejectButton) додається обробник події. У разі натискання вона скидає всі поля форми за допомогою методу reset і приховує форму через функцію hideAddMemberCard, завершуючи процес редагування.

Рисунок 15. Заповнення полів при редагуванні

Для заповнення з масиву activePersonsData знаходиться об’єкт мешканця за переданим memberId. Дані цього об’єкта, такі як прізвище, ім’я, факультет, номер телефону, тип та сума оплати, а також додаткова інформація, вставляються у відповідні поля форми. Це дозволяє користувачеві побачити поточні дані мешканця перед їх редагуванням. Для форми (personForm) додається обробник події submit, який забороняє стандартну поведінку (перезавантаження сторінки) та формує об’єкт formData з оновленими даними.

У цьому об’єкті дані про мешканця зчитуються з полів форми і додається ідентифікатор мешканця (memberId) разом з ідентифікатором кімнати (roomId) для подальшої передачі в функцію handleMemberButtonClick.  
Після формування об’єкта formData викликаються метод reset для очищення полів форми та функція hideAddMemberCard, яка приховує форму. У результаті функція повертає об’єкт formData для подальшого надсилання на сервер.

Функція editMemberInformation на мові JS:

    function editMemberInformation(memberId){

        openAddMemberCard();

        const rejectButton = document.getElementById('rejectbutton');

        rejectButton.addEventListener('click', function (event) {

            event.preventDefault();

            form.reset();

            hideAddMemberCard();

        });

        console.log('apd -----',activePersonsData);

        const person = activePersonsData.find(i => i.id === memberId);

        console.log('person',person)

        document.getElementById('lastName').value = person.lastName;

        document.getElementById('firstName').value = person.firstName;

        document.getElementById('fatherName').value = person.fatherName;

        document.getElementById('faculty').value = person.faculty;

        document.getElementById('phoneNumber').value = person.phoneNumber;

        document.getElementById('payType').value = person.payType;

        document.getElementById('paySum').value = person.payType

        document.getElementById('additionalInformation').value = person.additionalInformation;

        const form = document.getElementById('personForm');

        form.addEventListener('submit', async (event) => {

            event.preventDefault();

            const formData = {

                person: {

                    lastName: document.getElementById('lastName').value,

                    firstName: document.getElementById('firstName').value,

                    fatherName: document.getElementById('fatherName').value,

                    faculty: document.getElementById('faculty').value,

                    phoneNumber: document.getElementById('phoneNumber').value,

                    payType: document.getElementById('payType').value,

                    paySum: document.getElementById('paySum').value,

                    additionalInformation: document.getElementById('additionalInformation').value

                },

                memberId: memberId,

                roomId: roomId

            };

            form.reset();

            hideAddMemberCard();

            return formData;

        });

    }

Функція дозволяє користувачеві редагувати дані про мешканця безпосередньо в інтерфейсі вебзастосунку. Вона інтегрує поточну інформацію у форму, забезпечуючи її зручне оновлення, і формує структуровані дані для подальшої обробки. Завдяки цьому процес редагування є інтуїтивно зрозумілим та ефективним.

3.4 Тестування і результати

Для забезпечення коректної роботи вебзастосунку GurtKEY було проведено комплексне тестування, яке включало різні аспекти функціональності, продуктивності та безпеки. Метою тестування було виявлення можливих недоліків у системі, перевірка її відповідності вимогам і забезпечення стабільної роботи в реальних умовах експлуатації.

Функціональне тестування

Функціональне тестування було спрямоване на перевірку реалізованих функцій вебзастосунку відповідно до технічних вимог. Особливу увагу приділено роботі з основними модулями системи:

* **Візуалізація стану кімнат**: було перевірено правильність роботи кольорового кодування для кожного стану кімнати (вільна, частково зайнята, повністю зайнята, неактивна). Тестування включало динамічне оновлення стану після змін у базі даних.
* **Управління мешканцями**: перевірено функції додавання, редагування, переселення та виселення мешканців, а також відповідність виведених даних їх актуальному стану.
* **Навігація та інтерфейс**: було перевірено коректність роботи кнопок вибору поверхів, інтерактивності карток кімнат і доступності додаткових функцій для кожного елемента.

Тестування продуктивності

Для оцінки продуктивності вебзастосунку було проведено навантажувальне тестування з імітацією одночасної роботи кількох користувачів. Основні сценарії включали:

* Масове виконання запитів до бази даних для отримання інформації про кімнати.
* Одночасне додавання нових мешканців та оновлення інформації.
* Роботу з великою кількістю даних, включаючи переключення між поверхами та оновлення стану кімнат.

Результати тестування продемонстрували, що система здатна витримувати значні навантаження, забезпечуючи швидкий час відгуку навіть при роботі великої кількості користувачів одночасно.

Тестування безпеки

Особлива увага приділялася перевірці безпеки, оскільки система оперує чутливими даними про мешканців гуртожитку. Було виконано такі перевірки:

* **Захист бази даних**: протестовано автентифікацію запитів через API-ключі, що запобігає несанкціонованому доступу.
* **Обмеження доступу**: перевірено відповідність користувацьких прав доступу до різних функцій системи.
* **Безпечне зберігання даних**: проаналізовано механізми передачі та збереження інформації, що гарантує її цілісність і конфіденційність.
* **Захист від загроз**: протестовано стійкість системи до SQL-ін’єкцій, XSS-атак та інших поширених типів кіберзагроз.

Результати тестування

Результати комплексного тестування підтвердили, що вебзастосунок GurtKEY функціонує стабільно та відповідає заданим технічним і функціональним вимогам. Усі основні модулі працюють коректно, інтерфейс інтуїтивно зрозумілий, а час відгуку системи залишається в межах допустимих значень навіть при високих навантаженнях. Система успішно захищає дані мешканців, що гарантує її надійність і безпеку для реального використання.

#### 3.5 Використання в реальних умовах

На даний момент система GurtKEY демонструє високий рівень функціональності та стабільності, проте вона ще не повністю готова до практичного впровадження в умовах університету. Для реалізації повного спектра завдань, необхідних для ефективного управління гуртожитками, системі бракує кількох ключових функцій.

По-перше, відсутня можливість управління декількома гуртожитками одночасно. Це суттєве обмеження для університетів із розгалуженою структурою, де необхідно централізовано керувати поселенням і ресурсами кількох об'єктів.

По-друге, система поки що не включає функціонал для повноцінного контролю за оплатою. Інтеграція механізмів обліку платежів та звітності є важливим компонентом, який забезпечив би автоматизацію фінансових процесів і підвищив би зручність використання системи для адміністрації.

По-третє, не реалізовано інструменти для відстеження руху майна. Управління матеріальними ресурсами, такими як меблі та обладнання, є невід'ємною частиною ефективного адміністрування гуртожитків.

Попри ці недоліки, система GurtKEY може слугувати гарною базою для подальшої розробки та вдосконалення. Її модульна структура і гнучкість дозволяють легко додавати новий функціонал, адаптувати систему до змінних умов і вимог. Надалі, з розширенням можливостей, система може стати не лише ефективним інструментом для університетів, але й бути інтегрованою в більш широку екосистему управління студентським житлом.

Таким чином, хоча GurtKEY ще не готова до повномасштабного використання в реальних умовах, вона має значний потенціал для розвитку і, можливо, в майбутньому стане загальновизнаним рішенням у цій сфері.

**ВИСНОВКИ**

У результаті виконання кваліфікаційної роботи було створено прототип вебзастосунку GurtKEY для автоматизації процесів поселення у студентських гуртожитках. Вебзастосунок дозволяє здійснювати інтерактивний облік стану кімнат, додавати та редагувати дані про мешканців, а також генерувати візуальні звіти щодо заповненості гуртожитку.

Проведене тестування підтвердило відповідність розробленого програмного забезпечення функціональним вимогам, зокрема, візуалізація стану кімнат, інтерактивний вибір поверху та управління даними мешканців. Система показала стабільну роботу та зручність у використанні для кінцевих користувачів.

Робота також виявила низку напрямків для вдосконалення. Система ще не повністю готова до широкомасштабного впровадження, оскільки їй бракує таких функцій, як управління кількома гуртожитками, контроль за оплатою та рухом майна. Однак GurtKEY уже закладає надійну основу для подальшого розвитку, і його адаптація до практичних потреб університету є перспективним завданням.

Таким чином, розроблений вебзастосунок є сучасним і дієвим рішенням, яке може стати частиною цифрової трансформації управління гуртожитками, підвищуючи ефективність їхнього адміністрування та покращуючи якість обслуговування студентів.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. **Стовбуха Р.С.** Оптимізація веб-інтерфейсу для візуалізації поселення у гуртожитку: порівняльний аналіз розмітки та вибір оптимального підходу – Ніжин : Ніжинський державний університет імені Миколи Гоголя, 2024.
2. **Стовбуха Р.C.** Використання Webflow для верстки сайтів: огляд платформи, технічні можливості, переваги та практичні кейси Ніжин: Ніжинський державний університет імені Миколи Гоголя, 2024.
3. **Лазакович Є.В.** Розробка інформаційної системи обліку поселення в гуртожиток підходу – Ніжин : Ніжинський державний університет імені Миколи Гоголя, 2024.
4. Федорчук, М. І. Автоматизація процесів управління студентськими гуртожитками: навчальний посібник. — К.: Видавництво КНУ, 2018. — 256 с.
5. Бакуменко, О. В. Веб-технології для розробки інформаційних систем управління. — Харків: ХНУ, 2017. — 182 с.
6. Левченко, О. І. Інформаційні системи для управління студентським житлом. — Одеса: ОНУ, 2019. — 145 с.
7. W3C. Web Content Accessibility Guidelines (WCAG). URL: <https://www.w3.org/WAI/WCAG21/> (дата звернення: 03.05.2024).
8. Веб-технології. HTML5, CSS3, JavaScript. Підручник для розробників. — Львів: ЛДУ, 2016. — 320 с.
9. MDN Web Docs. JavaScript Guide. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide> (дата звернення: 09.05.2024).
10. Бондаренко, І. В. Основи інформаційних систем: підручник для студентів ВНЗ. — Київ: Академвидав, 2020. — 348 с.
11. Калашник, М. В. Технології веб-розробки: навчальний посібник. — Харків: ХНТУ, 2019. — 225 с.
12. Веб-застосунки на основі JavaScript: навчальний посібник / за ред. Л. М. Чернишова. — Київ: Вища школа, 2021. — 256 с.
13. Барон, С. С. Основи програмування для веб-застосунків. — Одеса: ОДУ, 2017. — 194 с.
14. W3C. HTML5 Specification. URL: <https://www.w3.org/TR/html5/> (дата звернення: 02.05.2024).
15. Turing, A. (2019). Web Application Architecture: Principles and Practice. Pearson Education.