Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту

Завідувач кафедри

_____Олена ОЛЬХОВСЬКА

(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ НА БАЗІ TELEGRAM-БОТА ДЛЯ ОРГАНІЗАЦІЇ ПОБУТОВИХ ПРОЦЕСІВ У СТУДЕНТСЬКОМУ ГУРТОЖИТКУ»
зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавр

Виконавець роботи Сакун Дмитро Юрійович

_____«__»_____202_р.

(підпис)

Науковий керівник к.ф.-м.н.,доц.,_____

_____«__»_____202_р.

(підпис)

Рецензент

ПОЛТАВА 2026

РЕФЕРАТ

Пояснювальна записка: 56 с., 4 рис., 1 табл., 2 додатки, 20 джерел.

КЛЮЧОВІ СЛОВА: TELEGRAM-БОТ, WEB-APP, АВТОМАТИЗАЦІЯ ПРИЛАДІВ, СІТКОВЕ ПЛАНУВАННЯ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ФОНОВІ ЗАВДАННЯ, ВАЛІДАЦІЯ ДАНИХ.

Об’єкт дослідження – процеси спільного використання, координації та взаємодії користувачів під час розподілу обмежених санітарно-побутових ресурсів у студентських гуртожитках закладу вищої освіти.

Мета роботи – проєктування, програмна реалізація та практичне впровадження розподіленої інформаційної системи на базі Telegram-бота та інтегрованого веб-додатка для автоматизації, оптимізації та безконфліктного календарного планування процесів бронювання часу використання побутової інфраструктури гуртожитку.

Методи дослідження – методи системного аналізу предметної області, об'єктно-орієнтованого моделювання процесів за допомогою уніфікованої мови UML, парадигми асинхронного та компонентного програмування. Інструментальними засобами розробки та реалізації системи обрано мову програмування Python [15] (фреймворки Django, Django REST Framework та Aiogram), реляційну СУБД PostgreSQL, брокер повідомлень RabbitMQ та чергу асинхронних завдань Celery. Клієнтський інтерфейс побудовано з використанням мови JavaScript, фронтенд-бібліотеки React та HTTP-клієнта Axios. Інфраструктурне розгортання та безпека середовища забезпечені технологіями контейнеризації Docker та криптографічного наскрізного шифрування Let's Encrypt.

Результати та їх новизна – проєктувано та програмно реалізовано оригінальну інформаційну систему «Пралка», яка забезпечує кросплатформений доступ користувачів до інтерактивного табло бронювання слотів. Розроблено та впроваджено алгоритми динамічного контролю добових і тижневих лімітів записів на основі чотирирівневої рольової моделі, модулі

жорсткої регулярної валідації кириличного написання імен, захисту від створення дублікатів профілів сторонніми особами та механізми негайного інфраструктурного блокування несправних пристроїв із червоним маркуванням у веб-інтерфейсі. Інтегровано модуль монетизації та обробки рекурентних платежів LiqPay з автоматичним приховуванням маркетингових акцій та верифікацією відповідності ПІБ через платіжні системи Apple Pay та Google Pay. Система успішно пройшла дослідну експлуатацію в гуртожитку №2 Полтавського університету економіки і торгівлі, де залучено 267 активних користувачів та стабільно оброблено 1982 бронювання, що дозволило повністю ліквідувати черги та підвищити побутовий комфорт для 82% опитаних мешканців.

Рекомендації щодо використання результатів роботи – розроблений програмний комплекс є повністю апаратно-незалежним, функціонально завершеним та рекомендованим до безпосереднього інфраструктурного розгортання в інших побутових кімнатах студентського кампусу ПУЕТ (зокрема, в гуртожитках №1 та №4). Перспективним напрямом використання є адаптація та масштабування архітектури коду для запуску сервісу в гуртожитках інших закладів вищої освіти регіону, насамперед у Полтавському державному аграрному університеті (ПДАУ).

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів та термінів
ГО	Громадська організація
ПУЕТ	Полтавський університет економіки і торгівлі
ПДАУ	Полтавський державний аграрний університет
СУБД	Система управління базами даних
API	Інтерфейс прикладного програмування (Application Programming Interface)
Axios	Спеціалізована клієнтська бібліотека для виконання асинхронних HTTP-запитів
Celery	Розподілена черга асинхронних завдань для виконання фонових процесів у реальному часі
Celery Beat	Вбудований планувальник завдань Celery для запускання процесів за регулярним розкладом
Django	Високорівневий веб-фреймворк на мові Python для розробки серверної логіки
DOM	Об'єктна модель документа (Document Object Model)
Docker	Платформа контейнеризації для ізоляції та автоматизації розгортання програмних модулів
DRF	Розширення Django REST Framework для швидкого проєктування захищених REST-ендпоінтів
FSM	Стан кінцевого автомата (Finite State Machine) для покрокової обробки вхідних даних
HTTP / HTTPS	Протокол передачі гіпертексту / Захищена версія протоколу з підтримкою шифрування трафіку
ID	Унікальний цифровий ідентифікатор користувача або об'єкта (Identifier)

IoT	Інтернет речей (Internet of Things)
JSON	Текстовий формат обміну даними, що базується на JavaScript (JavaScript Object Notation)
Let's Encrypt	Некомерційний центр сертифікації, що надає безкоштовні SSL/TLS-сертифікати для шифрування
LiqPay	Відкритий платіжний API-шлюз ПриватБанку для обробки онлайн-платежів та підписок
MQTT	Спрощений мережевий протокол передачі повідомлень на основі шаблону «публікація-підписка»
Nginx	Високопродуктивний веб-сервер та зворотний проксі-сервер (Reverse Proxy)
ORM	Технологія об'єктно-реляційного відображення баз даних у коді (Object-Relational Mapping)
PostgreSQL	Об'єктно-реляційна система управління базами даних зі строгим дотриманням принципів ACID
Python	Об'єктно-орієнтована асинхронна мова програмування високого рівня
RabbitMQ	Програмний брокер повідомлень асинхронної маршрутизації завдань між сервісами
React	Фронтенд-бібліотека мови JavaScript для побудови динамічних користувацьких інтерфейсів
REST	Архітектурний стиль взаємодії компонентів інформаційної системи (Representational State Transfer)
SPA	Односторінковий веб-додаток (Single Page Application)
SSL / TLS	Криптографічні протоколи безпеки для захисту комунікації в комп'ютерних мережах
UML	Уніфікована мова моделювання для графічної візуалізації архітектури та процесів ПЗ
UX	Досвід взаємодії користувача з інтерфейсом системи (User Experience)
WebApp	Інтегрований у месенджер веб-додаток, що функціонує всередині контейнера WebView

Зміст

ВСТУП

Впровадження сучасних інформаційних технологій у житлово-комунальну та соціальну сфери є одним із пріоритетних напрямів цифрової трансформації суспільства. Студентські гуртожитки закладів вищої освіти представляють собою специфічні соціо-комунальні екосистеми з високою концентрацією мешканців, де хаотичне спільне використання обмежених побутових ресурсів, зокрема пральних машин, неминуче призводить до виникнення побутових конфліктів, нераціональних витрат часу студентів та критичних пікових навантажень на локальні електричні мережі. В умовах сучасного дефіциту генеруючих потужностей в Україні автоматизація та рівномірне впорядкування графіків енергоспоживання стає особливо гострою технічною та організаційною задачею. Традиційні паперові методи контролю черги є морально застарілими та неефективними, що зумовлює високу актуальність розробки спеціалізованих апаратно-незалежних програмних комплексів для дистанційного календарного планування та менеджменту часу.

Об'єктом дослідження є процеси спільного використання та розподілу обмежених санітарно-побутових ресурсів у студентських гуртожитках.

Предметом дослідження виступають архітектурні моделі, алгоритми контролю лімітів та програмні засоби розробки розподілених інформаційних систем на базі месенджерів.

Метою кваліфікаційної роботи є проєктування, програмна реалізація та практичне впровадження розподіленої інформаційної системи на базі Telegram-бота та інтегрованого веб-додатка для автоматизації та безконфліктного планування процесів бронювання часу використання побутової інфраструктури гуртожитку.

Для досягнення поставленої мети було розв'язано задачі аналізу існуючих комерційних аналогів, проєктування клієнт-серверної архітектури з використанням мови UML [1], а також програмної реалізації системи з використанням сучасного та надійного стеку технологій. Розроблене серверне

ядро базується на веб-фреймворку Django, базі даних PostgreSQL [2] та асинхронному чат-боті на базі бібліотеки Aiogram [3] , тоді як графічний інтерфейс реалізовано за допомогою фронтенд-платформи React Web-App [4]. Для автоматизації тригерних сповіщень та рекурентних платіжних процесів інтегровано фонові обробники черг Celery [5] і RabbitMQ [6], а вся інфраструктура розгорнута в контейнерах Docker [7] під захистом SSL-сертифікатів Let's Encrypt. Окремо розроблено унікальні алгоритми багатоетапної валідації користувачів, жорсткого контролю кириличного написання імен, захисту від дублікатів акаунтів та часового блокування несправних пристроїв.

Практичне значення отриманих результатів полягає у створенні готового до експлуатації програмного продукту «Пралка», який успішно пройшов підконтрольну апробацію на базі гуртожитку №2 Полтавського університету економіки і торгівлі. Впровадження системи дозволило залучити 267 активних користувачів та стабільно обробити 1982 бронювання при повній відсутності технічних збоїв, забезпечивши при цьому ліквідацію черг для 82% опитаних мешканців. Інтеграція платіжного шлюзу LiqPay [8] підтвердила фінансову самоокупність проєкту на користь громадської організації «Зелені Ери», а висока інфраструктурна гнучкість архітектури доводить перспективність її подальшого масштабування у гуртожитках №1 та №4 університету, а також впровадження платформи в інфраструктуру гуртожитків Полтавського державного аграрного університету.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

1.1. Опис предметної області: організація спільного використання ресурсів у студентських гуртожитках

Сучасний студентський гуртожиток закладу вищої освіти представляє собою специфічну соціо-комунальну екосистему, яка характеризується високою концентрацією мешканців на обмеженій житловій площі та спільним використанням об'єктів санітарно-побутової інфраструктури. Однією з найбільш критичних і чутливих зон у забезпеченні щоденного побутового комфорту студентів є організація процесу прання особистих речей. Обмежена кількість пральних машин, що встановлені в спеціалізованих побутових кімнатах гуртожитків, має обслуговувати сотні студентів, що неминуче призводить до виникнення організаційних та технічних проблем.

Традиційні підходи до управління чергою побутових кімнат, які базуються на принципах «живої черги» або веденні фізичних (паперових) журналів запису, на сучасному етапі розвитку інформаційних технологій є морально застарілими та неефективними. Паперові носії інформації піддаються швидкому зносу, фізичному пошкодженню та не дають можливості віддаленого контролю. Головними недоліками неавтоматизованого розподілу часу є:

- виникнення міжособистісних конфліктів між мешканцями через порушення черговості або непрозорість процесу запису;
- відсутність можливості дистанційного бронювання, що створює значні незручності для студентів, які поєднують навчання з працевлаштуванням або науковою діяльністю;
- нераціональне витрачання часового ресурсу здобувачів освіти, змушених особисто відвідувати побутові кімнати для перевірки статусу пральних машин.

З огляду на це, виникає гостра потреба у впровадженні віртуальних інструментів менеджменту часу, які забезпечують справедливий розподіл

ресурсів побутового призначення. Автоматизація процесу дозволяє підвищити загальний рівень комфорту проживання та оптимізувати щоденну рутину студентів.

Окремим важливим аспектом опису предметної області є екологічний та енергетичний чинники. В сучасних реаліях функціонування житлово-комунального сектору України, що супроводжується дефіцитом генеруючих потужностей та обмеженнями у постачанні електроенергії, хаотичне використання енергоємного обладнання (яким є пральні машини) створює надмірні пікові навантаження на локальну електричну мережу гуртожитку. Завдяки добросовісному та рівномірному плануванню часових слотів з'являється можливість уникнути критичних перевантажень інфраструктури, ліквідувати черги, а також суттєво заощадити природні та енергетичні ресурси.

Додатковою проблемою досліджуваної предметної області є верифікація користувачів та захист від несанкціонованого доступу сторонніх осіб. Оскільки автоматизована система оперує внутрішніми ресурсами конкретного навчального закладу чи гуртожитку, виникає технічна задача ідентифікації мешканців для унеможливлення реєстрації осіб, які не мають законного права на використання даного сервісу. Масштаби предметної області можна оцінити на прикладі одного студентського гуртожитку, де кількість активних користувачів, що одночасно потребують доступу до обмеженого числа пральних машин (зазвичай від 2 до 4 одиниць), становить понад 260 осіб. Це вимагає розробки гнучких динамічних лімітів, алгоритмів контролю за дотриманням розкладу та зручного графічного представлення інформації про зайнятість обладнання. Таким чином, предметна область вимагає створення комплексної інформаційної системи, що поєднує мобільну доступність, автоматизований контроль правил користування та інструменти адміністративного моніторингу.

1.2. Змістовна постановка задачі

Змістовна постановка задачі полягає у визначенні чітких меж функціональності проектованого програмного комплексу, структуризації вхідних та вихідних даних, а також у формулюванні бізнес-правил та обмежень, якими оперує інформаційна система «Пралка».

Основною метою розробки є створення розподіленої інформаційної системи для автоматизації процесів моніторингу, планування та бронювання часу використання побутового обладнання (пральних машин) мешканцями студентських гуртожитків. Система має забезпечити прозорий розподіл ресурсів, мінімізувати вплив людського фактору та ліквідувати черги.

Для досягнення поставленої мети інформаційна система повинна вирішувати комплекс взаємопов'язаних задач, які поділяються на кілька функціональних блоків:

1. Управління обліковими записами та рольова модель

Система має забезпечувати реєстрацію користувачів через месенджер Telegram із прив'язкою до унікального ідентифікатора (Telegram ID) та номеру телефону. Програмне забезпечення повинно підтримувати динамічне розмежування прав доступу на основі чотирьох базових статусів користувачів:

- Неверифікований користувач: початковий статус. Обмеження: можливість створення лише 1 запису на день (не більше 1 запису на тиждень), видимість графіку бронювання виключно на поточну добу. Повністю заблоковано доступ до надсилання скарг та сервісних повідомлень про несправність техніки.
- Верифікований користувач: статус, що надається після успішного підтвердження факту проживання у гуртожитку за допомогою завантаження фотокопії перепустки. Права: створення до 2 записів на день (але не більше 4 на тиждень), видимість графіку на поточний та наступний дні.

- Обмежені ліміти: штрафний статус, що присвоюється адміністратором за порушення правил. Обмеження: до 1 запису на день (не більше 2 на тиждень), видимість графіку на 1 день.
- Преміум-користувач: статус, який активується після оформлення платної підписки. Переваги: розширені ліміти (до 6 записів на тиждень, макс. 3 записи на один день), видимість графіку на 3 дні вперед, пріоритетна підтримка та доступ до функції відстеження звільнених часових слотів.

2. Менеджмент часових слотів та логічні обмеження

Ядром системи є інтерактивний графік, розбитий на фіксовані часові комірки (слоти) тривалістю в 1 годину. Процес бронювання підпорядковується суворим алгоритмічним обмеженням для запобігання зловживанням:

- заборонено бронювати часові слоти, тривалість яких за поточним часом сервера вже розпочалася або завершилася;
- скасування існуючого запису користувачем дозволяється лише до моменту фактичного настання забронюваного часу;
- система повинна автоматично перевіряти ПІБ користувачів під час реєстрації на предмет використання латинських символів (допускається лише кирилиця) та виявляти потенційні дублікати акаунтів.

3. Автоматизоване сповіщення та тайм-менеджмент

Для координації дій користувачів та підтримки дисципліни використання обладнання система має функціонувати в режимі реального часу та забезпечувати тригерну розсилку повідомлень:

- надсилання нагадувань про заплановане прання за 1 годину та за 5 хвилин до початку обраного слоту;
- надсилання сповіщення про необхідність вивантаження речей та звільнення пральної машини за 5 хвилин до завершення часу броні;
- автоматичне щоденне (а в подальшому — раз на 7 днів) нагадування неверифікованим користувачам про обмеження їхнього функціоналу та необхідність пройти верифікацію профілю.

4. Модуль монетизації (Преміум-підписка)

Інформаційна система повинна містити інтегрований фінансовий шлюз для обробки регулярних платежів (рекурентних транзакцій) на користь ГО «Зелені Ери» у розмірі 30 грн/місяць. Модуль має підтримувати:

- можливість надання маркетингової знижки (акційної ціни, наприклад, 1 грн) на перший місяць використання;
- автоматичне приховування акційних пропозицій після першої успішної оплати;
- механізм безперешкодного скасування підписки через інтерфейс бота за командою /subscribe_cancel;
- верифікацію відповідності ПІБ користувача даним, отриманим через електронні платіжні системи (Apple Pay / Google Pay).

5. Модуль адміністрування та моніторингу інфраструктури

Для диспетчеризації та контролю роботи сервісу представникам адміністрації мають надаватися такі можливості:

- верифікація нових користувачів через спеціалізовану закриту Telegram-групу за допомогою інтерактивних кнопок швидкої дії («Ок» / «Видалити акаунт» із зазначенням причини видалення);
- ручне блокування окремих пральних машин у разі технічної несправності на заданий проміжок часу (від / до) із відповідним візуальним маркуванням (червоним кольором) у веб-інтерфейсі для унеможливлення вибору цих слотів студентами;
- повне або тимчасове блокування користувачів-порушників із виведенням індивідуального візуального пояснення причини блокування в інтерфейсі бота;
- таргетована розсилка інформаційних повідомлень за гнучкими фільтрами (усім користувачам, конкретній особі, за номером гуртожитку, за статусом верифікації чи наявністю преміум-підписки).

Вхідні та вихідні дані системи

Вхідними даними системи є:

- реєстраційні відомості користувача (ПІБ кирилицею, номер телефону, унікальний Telegram ID);

- графічні дані для верифікації (фотокопія або скан-копія дійсної перепустки до гуртожитку);
- параметри конфігурації інфраструктури (список доступних пральних машин, номери гуртожитків, часові інтервали);
- платіжні токени, що надходять від стороннього платіжного шлюзу (LiqPay) після проведення транзакції;
- сервісні команди користувача (/support, /subscribe_cancel) та запити на бронювання конкретної комірки графіку.

Вихідними даними системи є:

- інтерактивна веб-матриця (графік) зайнятості та доступності слотів у розрізі кожної пральної машини з відображенням позначок верифікації користувачів;
- сервісні текстові та тригерні сповіщення (нагадування, попередження про ліміти, сповіщення про успішне або скасоване бронювання, фінансові квітанції);
- адміністративні картки модерації користувачів та логи збігів (дублікатів) імен в адмін-панелі;
- аналітичні звіти щодо інтенсивності використання обладнання (кількість успішних бронювань, статистика підписок) для керівництва ГО «Зелені Ери» та адміністрації ЗВО.

1.3. Специфікація бізнес-вимог та функціональних обмежень системи

Для забезпечення стабільного функціонування, виключення зловживань та оптимізації навантаження на апаратно-програмну частину комплексу «Пралка», розроблено строгу специфікацію бізнес-вимог (Business Requirements) та функціональних обмежень (Functional Constraints). Володільцем персональних даних та адміністратором сервісу є громадська організація «Зелені Ери». Програмний продукт надає користувачам виключно обмежене, невиключне право використання без переходу прав інтелектуальної власності.

Нижче наведено деталізований опис логічних правил, лімітів та бізнес-процесів системи.

1. Функціональні обмеження рівнів доступу (статусів користувачів)

Доступ до інтерфейсу запису, що функціонує через Telegram-бот та веб-сторінку, диференціюється за статусами. Кожен статус накладає жорсткі обмеження на кількість бронювань та глибину відображення графіку:

- Неверифікований користувач: встановлюється автоматично після первинної реєстрації за номером телефону месенджера Telegram. Користувачу доступний 1 запис на день, але не більше 1 запису на тиждень. Видимість графіку обмежена лише поточною добою (1 день видимості). Такі користувачі позбавлені права залишати скарги чи повідомляти про несправності пральних машин.
- Верифікований користувач: присвоюється мешканцям гуртожитку, які проживають на території дії сервісу та надали дійсну перепустку для підтвердження особи. Статус дозволяє здійснювати 2 записи на день, але не більше 4 записів на тиждень. Глибина видимості графіку розширюється на 2 дні (поточний та наступний за ним день).
- Обмежені ліміти (Штрафний статус): тимчасовий статус, що призначається адміністратором вручну за порушення внутрішніх правил користування сервісом. Передбачає зниження лімітів до 1 запису на день (не більше 2 на тиждень) та видимість графіку лише на поточний день.
- Преміум-користувач: активується за умови оформлення платної підписки. Надає право на 6 записів на тиждень (але не більше 3 за один день), розширює видимість графіку на 3 дні вперед, відкриває доступ до функції відстеження місць, що звільнилися, та пріоритетної підтримки.

2. Матриця порівняння лімітів та можливостей

3. Специфікація бізнес-правил монетизації

Фінансовий модуль системи регулюється такими суворими вимогами:

- Тарифікація: стандартна вартість Преміум-Підписки становить 30 гривень на місяць.

- Рекурентність: оплата списується автоматично за кожен наступний період (щомісячно) через інтегровані платіжні системи третіх сторін без збереження повних реквізитів карток на серверах адміністратора.
- Маркетингові акції: підтримується можливість оформлення підписки на перший місяць за зниженою (акційною) ціною (наприклад, 1 гривня). Акційна кнопка автоматично видаляється з інтерфейсу користувача відразу після проведення першої транзакції.
- Процедура скасування: скасування підписки реалізовано через введення текстової команди /subscribe_cancel. Наступні автоматичні списання припиняються негайно, проте користувач зберігає преміум-статус до кінця вже оплаченого періоду.

Для наочності функціональні обмеження зведені у порівняльну таблицю 1.1.

- Таблиця 1.3.1. Функціональні обмеження

Параметр обмеження	Неверифікований	Верифікований	Обмежені ліміти	Преміум-акаунт
Добовий ліміт записів	1 слот	2 слоти	1 слот	До 3 слотів
Тижневий ліміт записів	1 слот	4 слоти	2 слоти	6 слотів
Глибина видимості графіку	1 день	2 дні	1 день	3 дні
Функція «Пошук вільних місць»	Ні	Ні	Ні	Так
Доступ до подання скарг	Ні	Так	Так	Так (пріоритет)

4. Системні обмеження безпеки та валідації даних

Для упередження технічних збоїв та зловживань у системі діють такі обмеження:

- Часова валідація слотів: система блокує можливість запису на часову комірку, якщо її календарний час за сервером уже розпочався. Користувач не може скасувати свій запис, якщо тривалість цього слоту вже настала.
- Модерація фейкових профілів: при реєстрації нових користувачів дані дублюються в закриту Telegram-групу модерації з інтерактивними кнопками «Ок» та «Видалити акаунт». Кнопка видалення дозволяє

надіслати користувачу сервісне повідомлення із зазначенням причини відмови.

- Захист від дублікатів та мовний контроль: впроваджено автоматичний пошук збігів за прізвищем у базі даних адміністратора. Під час заповнення профілю встановлено заборону на використання латинських символів у полі імені — система приймає виключно кириличний набір літер.
- Апаратне блокування: у разі виходу пральної машини з ладу адміністратор має технічну можливість заблокувати пристрій на певний період (від / до). У веб-інтерфейсі WebApp відповідний стовпчик маркується червоним кольором, що робить його неактивним для взаємодії мешканців.
- Тригерні розсилки: система підтримує сегментовані розсилки через Celery Beat за фільтрами: статус підписки, номер гуртожитку, статус верифікації. Для неверифікованих акаунтів діє алгоритм автоматичного нагадування: перше повідомлення надходить наступного дня після реєстрації, а повторні — кожні 7 днів до моменту завантаження документів.
- Шифрування даних: усі комунікаційні процеси, передача платіжних токенів LiqPay та завантаження фотокопій документів здійснюються виключно через безпечні протоколи з використанням криптографічних SSL-сертифікатів Let's Encrypt.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Аналіз існуючих рішень для дистанційного керування побутовими процесами та аналогічного програмного забезпечення

Проблема автоматизації та координації колективного використання обмежених побутових ресурсів не є унікальною і лежить на перетині технологій інтернету речей (IoT), систем управління часом (Time Management Systems) та концепцій «розумного міста» (Smart City). Для вирішення подібних задач у світовій та вітчизняній практиці застосовуються різноманітні програмні та апаратні архітектурні підходи. В межах аналізу існуючих рішень доцільно класифікувати аналогічне програмне забезпечення за трьома основними категоріями.

Категорія 1. Пропрієтарні екосистеми виробників побутової техніки

Великі міжнародні корпорації (такі як LG, Samsung, Bosch, Miele) активно розвивають власні програмні платформи для дистанційного моніторингу та керування побутовими приладами:

- LG ThinQ та Samsung SmartThings: мобільні застосунки, які дозволяють користувачам віддалено запускати цикли прання, отримувати Push-сповіщення про завершення процесу та відстежувати енергоспоживання пристроїв через хмарні сервіси.
- Home Connect (Bosch/Siemens): відкрита партнерська екосистема, що надає API для інтеграції побутової техніки із зовнішніми системами автоматизації («розумний будинок»).

Головним обмеженням цієї категорії рішень є орієнтація на *індивідуальне*, а не колективне використання. Вони не містять вбудованих механізмів гнучкого календарного планування для сотень незалежних користувачів, не підтримують рольові моделі доступу для студентських спільнот та вимагають наявності специфічного, дороговартісного обладнання з інтегрованими Wi-Fi модулями, що є економічно недоцільним для більшості державних ЗВО.

Категорія 2. Спеціалізовані комерційні платформи для пральних самообслуговування (Laundromats)

У країнах Західної Європи та США для управління пральними кімнатами у студентських кампусах та комерційних пральнях самообслуговування використовуються спеціалізовані B2B-платформи:

- Circuit Laundry: популярна у Великій Британії система, яка поєднує мобільний застосунок та хмарну панель керування. Користувачі можуть бачити статус зайнятості машин у реальному часі, резервувати їх та здійснювати оплату за кожне прання через інтегровані платіжні шлюзи.
- LaundryView (CSC ServiceWorks): комплексна система моніторингу, яка візуалізує завантаженість пральних залів кампусів, надсилає текстові сповіщення користувачам та надає адміністраторам детальну аналітику використання обладнання.

Такі системи є найбільш близькими за своїм функціональним призначенням до об'єкту розробки. Проте їх впровадження в українських студентських гуртожитках ускладнене високою вартістю ліцензійного програмного забезпечення, необхідністю закупівлі спеціалізованих апаратних контролерів для підключення до кожної пральної машини, а також відсутністю адаптації під специфічні внутрішні вимоги українських ЗВО (наприклад, жорстку верифікацію за локальними документами та динамічне обмеження лімітів без стягнення плати за кожне окреме прання).

Категорія 3. Загальні інструменти тайм-менеджменту та календарного планування

Для вирішення задачі розподілу часу часто використовуються універсальні хмарні календарі та платформи для організації спільної роботи:

- Google Календар / Teamup: безкоштовні або умовно-безкоштовні сервіси, де кожна пральна машина створюється як окремий «ресурс» або календар. Користувачі можуть самостійно створювати події, займаючи певні часові інтервали.

Даний підхід приваблює відсутністю необхідності розробки власного програмного коду, проте має критичні недоліки в умовах гуртожитку. Загальні

календарі не дозволяють реалізувати автоматичний контроль складних бізнес-правил (наприклад, заборону використання латини в іменах, тригерні тижневі ліміти залежно від статусу верифікації користувача, автоматичне виявлення дублікатів акаунтів). Крім того, інтерфейс таких систем не оптизований під швидку взаємодію через месенджери та створює високий ризик випадкового видалення або редагування чужих записів через відсутність жорсткого розмежування прав доступу на рівні окремих слотів.

Для систематизації результатів огляду аналогів проведено порівняльний аналіз архітектурних характеристик існуючих рішень (таблиця 2.1).

Таблиця 2.1. Порівняння існуючих рішень

Критерій порівняння	Пропрієтарні IoT-додатки	Комерційні платформи (кампусні)	Загальні хмарні календарі
Архітектура інтерфейсу	Мобільний додаток (Native)	Мобільний додаток + Веб	Веб-інтерфейс / Додаток
Орієнтація на користувача	Індивідуальна	Колективна (комерційна)	Колективна (універсальна)
Апаратна незалежність	Повністю відсутня	Відсутня (потрібні модулі)	Повна
Гнучкий контроль лімітів	Відсутній	Частковий (через оплату)	Відсутній
Специфічна верифікація	Відсутня	Відсутня	Відсутня
Доступність розробки	Закрите ПЗ (API обмежене)	Закрите комерційне ПЗ	Відкрите хмарне ПЗ

Таким чином, проведений аналіз підтверджує, що жодне з існуючих рішень не здатне повною мірою задовольнити специфічні організаційні, технічні та фінансові вимоги до автоматизації побутових процесів у вітчизняних студентських гуртожитках, що обґрунтовує доцільність розробки власної інформаційної системи.

2.2. Позитивні аспекти та переваги відомих систем

Детальний аналіз наявного ринку програмного забезпечення та спеціалізованих платформ автоматизації дозволяє виділити низку архітектурних, функціональних та інтерфейсних рішень, які демонструють високу ефективність і можуть бути визнані сучасними індустріальними стандартами. Ці позитивні аспекти відомих аналогів охоплюють різні технологічні та експлуатаційні напрями, створюючи надійну теоретичну та практичну основу для проектування нових розподілених систем.

Однією з ключових переваг пропрієтарних екосистем, таких як LG ThinQ або Samsung SmartThings, а також комерційних кампусних платформ типу LaundryView, є забезпечення концепції наскрізної видимості стану обладнання в реальному часі. З інженерної точки зору це реалізується за допомогою архітектурних шаблонів, що базуються на подіях, та протоколів передачі даних із низькою затримкою, зокрема WebSocket або MQTT. Такий підхід дозволяє користувачам миттєво отримувати інформацію про поточний статус пристрою та точний час до завершення робочого циклу, що повністю ліквідує необхідність фізичного відвідування побутових приміщень для візуального контролю техніки.

Не менш важливим аспектом є функціонування подієво-орієнтованих систем асинхронних сповіщень, які активно використовуються у спеціалізованих комерційних рішеннях на кшталт Circuit Laundry. Вони демонструють високі показники задоволеності користувачів завдяки предиктивному та тригерному інформуванню, що включає автоматичне надсилання сервісних повідомлень за певний інтервал часу до початку запланованої сесії, сповіщення про завершення технологічного циклу або критичну зупинку обладнання. Це суттєво підвищує загальну дисципліну користувацької спільноти, оскільки оптимізує коефіцієнт корисного використання кожної одиниці техніки та мінімізує простій інфраструктури в гуртожитках.

У контексті дизайну інтерфейсів для задач тайм-менеджменту високі стандарти задають універсальні хмарні рішення, серед яких яскравими прикладами є Google Календар та Teamup. Візуалізація доступного ресурсу у

вигляді інтерактивної матриці, де стовпці представляють конкретні фізичні пристрої, а рядки — дискретні часові інтервали, є найбільш інтуїтивно зрозумілою для кінцевого користувача. Така ергономіка сіткових графіків планування забезпечує високу швидкість зчитування інформації про зайнятість ресурсів, вимагає мінімальної кількості дій для здійснення успішного бронювання та надає можливість швидкого візуального порівняння альтернативних варіантів часу у разі високої завантаженості системи.

Сучасні комерційні пральні системи також успішно реалізують фінансову інтеграцію через безпечні шлюзи та API платіжних систем. Позитивним досвідом у цій сфері є використання технологій токенізації, зокрема Apple Pay, Google Pay та карткових рекурентних платежів, які дозволяють користувачам здійснювати фінансові операції, такі як оплата послуг або придбання підписок, без необхідності ручного введення повних реквізитів банківських карток під час кожної транзакції. Це не лише підвищує конверсію платних функцій, а й гарантує повну відповідність жорстким вимогам безпеки даних індустрії платіжних карток PCI DSS.

Нарешті, відомі комерційні B2B-платформи надають операторам систем потужні інструменти для централізованого адміністрування та аналітики за допомогою хмарних панелей керування. Ці інструменти дозволяють агрегувати статистичні дані, гнучко та віддалено блокувати пристрої на технічне обслуговування, а також сегментувати користувацьку базу для надсилання таргетованих сповіщень. Комбінація всіх цих перевірених рішень утворює солідний архітектурний базис для проектування та впровадження вітчизняного програмного продукту, адаптованого до специфіки студентського середовища.

2.3. Вади, недоліки та труднощі в експлуатації оглянутих аналогів

Незважаючи на високий рівень технологічного розвитку та наявність значної кількості позитивних аспектів у сучасних архітектурних рішеннях, детальний критичний аналіз виявляє суттєві вади, недоліки та експлуатаційні труднощі, які унеможливлюють їх пряме впровадження в екосистему

студентських гуртожитків вітчизняних закладів вищої освіти. Головні проблеми наявних аналогів лежать у площині високої апаратної залежності, значної вартості ліцензування, відсутності гнучких механізмів автоматичного контролю за дотриманням лімітів та вразливості до деструктивного впливу людського фактору.

Розглядаючи пропрієтарні екосистеми інтернету речей від провідних міжнародних виробників побутової техніки, першочерговою вадою слід визнати їх повну прив'язку до конкретних брендів та моделей вищого цінового сегменту, що обов'язково обладнані інтегрованими модулями бездротового зв'язку. Державні заклади вищої освіти України, як правило, укомплектовують побутові кімнати гуртожитків стандартним комерційним або базовим промисловим обладнанням без підтримки вбудованих Smart-функцій. Крім того, архітектура таких мобільних додатків орієнтована виключно на індивідуальне або приватне сімейне використання. Це створює технічний тупик при спробі організувати одночасний паралельний доступ для сотень незалежних клієнтів, оскільки у подібному програмному забезпеченні повністю відсутні модулі календарного планування, черговості, а також інструменти адміністративного розмежування прав доступу мешканців.

Спеціалізовані комерційні платформи для кампусних пралень, хоча й розроблялися безпосередньо під задачу колективного планування, супроводжуються критичними фінансовими та інфраструктурними труднощами під час спроби їх адаптації та розгортання. Експлуатація таких закордонних систем вимагає значних капіталовкладень у придбання ліцензійного програмного забезпечення та спеціалізованих апаратних контролерів, які необхідно фізично інтегрувати у внутрішні електричні схеми кожної пральної машини для зчитування інформації про її стан. За умов обмеженого фінансування університетів та повної відсутності офіційної технічної підтримки таких компаній на ринку України, даний підхід є економічно та технічно нежиттєздатним. Додатковим недоліком є жорстка комерціалізація платформ, орієнтована на похвилинну оплату або тарифікацію кожного

окремого циклу прання, що суперечит некомерційній концепції підтримки студентів та вимагає складного юридичного оформлення фінансових потоків.

Використання універсальних хмарних календарів та безкоштовних платформ сумісного планування, у свою чергу, демонструє найвищий рівень вразливості до людського фактору та щоденних організаційних збоїв. Оскільки такі системи не мають жорстких програмованих обмежень бізнес-логіки на рівні окремих записів, користувачі отримують технічну можливість випадково або навмисно видаляти, переміщувати чи редагувати чужі часові слоти, що провокує міжособистісні конфлікти. Повна відсутність вбудованих засобів автоматичної валідації вхідних даних дозволяє здійснювати реєстрацію під вигаданими іменами, використовувати латинські символи замість кирилиці, створювати фейкові дублікати облікових записів, що повністю нівелює можливість контролю з боку адміністрації. Окрім цього, універсальні календарі не здатні забезпечити динамічний розрахунок лімітів залежно від статусу верифікації мешканця гуртожитку та позбавлені механізмів гнучкої тригерної розсилки сповіщень.

У підсумку, експлуатація оглянутих аналогів супроводжується значними труднощами: від високої вартості та апаратної несумісності до критичних прогалин у безпеці даних і неможливості автоматичного контролю внутрішніх правил спільноти. Існуючі програмні продукти не враховують специфіку організації побуту в українських гуртожитках, де першочерговим завданням є не монетизація кожної хвилини роботи техніки, а забезпечення справедливого, безконфліктного та безпечного розподілу обмеженого ресурсу часу між мешканцями. Зазначені недоліки підтверджують технічну та соціальну необхідність розробки індивідуального, апаратно-незалежного та інтегрованого у месенджер програмного комплексу, що поєднує строгу логіку бізнес-обмежень із високим рівнем UX-ергономіки.

2.4. Обґрунтування необхідності та актуальності розробки власного програмного продукту

Проведені дослідження сучасного ринку програмного забезпечення та критичний аналіз наявних аналогів свідчать про існування суттєвого технологічного та функціонального вакууму у сфері автоматизації спільного використання комунальних ресурсів студентських гуртожитків. Більшість існуючих комерційних та пропрієтарних систем або вимагають значних фінансових витрат на ліцензування та апаратну модернізацію, або є занадто вразливими до деструктивного впливу людського фактору через відсутність засобів гнучкого програмованого контролю внутрішніх правил. У зв'язку з цим виникає об'єктивна необхідність проектування та реалізації власного програмного продукту, який би поєднував у собі економічну доступність, повну апаратну незалежність, високу ергономіку користувацького інтерфейсу та суворий алгоритмічний контроль за дотриманням лімітів.

Актуальність розробки індивідуальної інформаційної системи «Пралка» першочергово обумовлена вибором її архітектурної моделі, що базується на гібридному поєднанні можливостей асинхронного Telegram-бота та інтегрованого реактивного веб-додатка (Web-App). Таке інженерне рішення дозволяє повністю розв'язати проблему кросплатформеної доступності без необхідності проектування, публікації та підтримки дорогих нативних мобільних додатків під різні операційні системи. Використання месенджера Telegram як первинної точки взаємодії забезпечує максимальне охоплення цільової аудиторії, оскільки це програмне забезпечення вже є базовим інструментом комунікації для переважної більшості студентів. Впровадження React-інтерфейсу безпосередньо у діалогове вікно чату дозволяє реалізувати динамічну, реактивну сітку часових слотів, що мінімізує кількість дій користувача для здійснення успішного бронювання та усуває проблему перевантаження екрану застарілими текстовими меню.

Особливу наукову та інженерну цінність власної розробки становить її логіко-алгоритмічне наповнення, повністю адаптоване під специфіку соціо-

комунальної структури студентського середовища. На відміну від універсальних хмарних календарів, проектована система впроваджує строгу чотирирівневу рольову модель доступу, яка гнучко регулює добові та тижневі ліміти записів і глибину видимості графіку залежно від статусу верифікації профілю. Впровадження інноваційних модулів автоматичного контролю вхідних даних — зокрема, заборони використання латинських символів у полях ідентифікації, механізму автоматичного пошуку дублікатів акаунтів, двофакторної модерації нових користувачів через закриту адмін-групу, а також блокування можливості створення або скасування записів на час, який уже фактично розпочався — забезпечує високий рівень кібербезпеки та ліквідує будь-які можливості для зловживань чи створення фейкових профілів сторонніми особами.

Економічна та організаційна доцільність проекту підсилюється його успішною інтеграцією з некомерційним сектором в особі громадської організації «Зелені Ери». Створення захищеного вбудованого модуля обробки рекурентних мікроплатежів на базі сучасного платіжного шлюзу LiqPay дозволяє забезпечити повну фінансову стійкість та самоокупність сервісу за рахунок підписок на розширений Преміум-функціонал, не накладаючи жодного фінансового чи юридичного навантаження на бюджет закладу вищої освіти. Система є повністю апаратно-незалежною, що дозволяє розгорнути її на базі вже існуючого в гуртожитку парку пральних машин без внесення змін до їхніх внутрішніх електричних схем, надаючи при цьому адміністраторам гнучкий інструмент швидкого часового блокування несправних пристроїв із миттєвим червоним маркуванням у веб-інтерфейсі для студентів.

Практична необхідність та своєчасність розробки підтверджується вагомими результатами її експериментальної апробації на базі гуртожитку №2 Полтавського університету економіки і торгівлі (ПУЕТ). Протягом періоду підконтрольної експлуатації розроблений серверний стек технологій на базі Python, Django, Celery та RabbitMQ продемонстрував абсолютну технічну стабільність та відсутність критичних архітектурних збоїв під час обробки високих потоків асинхронних запитів. За короткий проміжок часу система

залучила 267 активних користувачів із числа мешканців та успішно обробила 1982 бронювання слотів. Висока соціальна ефективність розробки, підтверджена результатами соціологічного моніторингу (понад 82% опитаних студентів офіційно засвідчили ліквідацію черг та значне підвищення побутового комфорту), а також наявність офіційних планів щодо масштабування та запуску системи в гуртожитках №1 і №4 ПУЕТ та гуртожитках Полтавської державної аграрної академії (ПДАУ) повністю доводять високу актуальність, практичну значущість та перспективність даної кваліфікаційної роботи.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

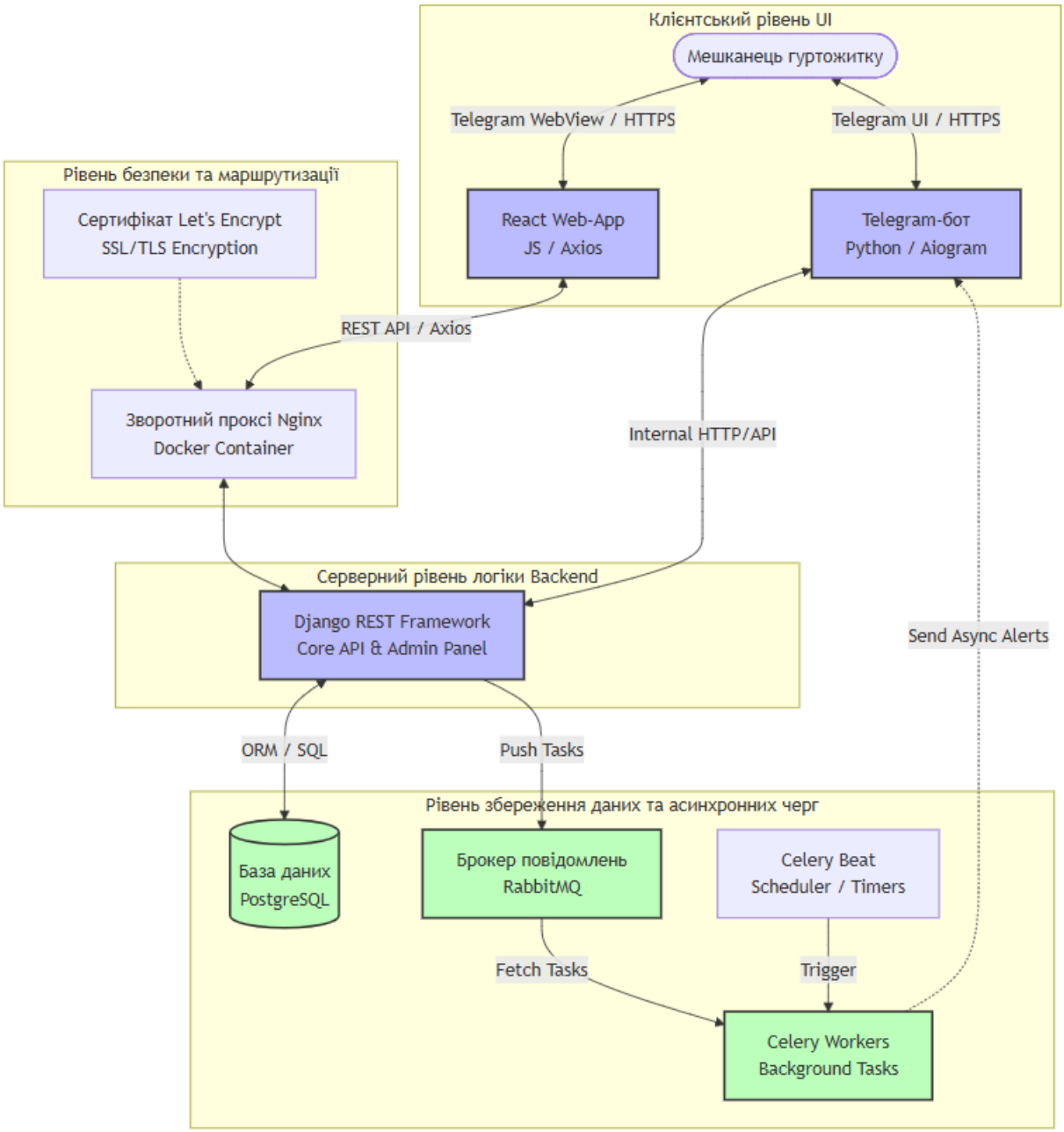
3.1. Проєктування архітектури інформаційної системи: клієнт-серверна модель (Telegram-бот та Web-App)

Архітектурне проєктування є фундаментальним етапом розробки будь-якої складної інформаційної системи, оскільки воно визначає структуру програмного комплексу, взаємозв'язки між його компонентами, а також забезпечує виконання вимог щодо масштабованості, надійності та безпеки даних. Для реалізації сервісу координації та автоматизації побутових процесів «Пралка» було обрано класичну багатокомпонентну клієнт-серверну архітектуру (Client-Server Architecture) [11]. Вибір цієї парадигми обумовлений необхідністю централізованого збереження інформації про стан обмежених ресурсів (пральних машин) та забезпечення одночасного, ізольованого та оперативного доступу до цих даних великої кількості географічно розподілених користувачів з різних мобільних та десктопних пристроїв.

Клієнтський рівень системи (Frontend layer) розроблено як гібридний користувацький інтерфейс, інтегрований в екосистему месенджера Telegram. Він складається з двох взаємопов'язаних технологічних компонентів: асинхронного чат-бота на базі фреймворку Aiogram та сучасного односторінкового веб-додатка (Single Page Application) на базі бібліотеки React, що функціонує за технологією Telegram WebApp. Telegram-бот виступає первинним шлюзом для взаємодії зі студентами, виконуючи функції автентифікації користувачів за номером телефону, збору первинних реєстраційних даних, обробки текстових команд підтримки та надсилання сервісних сповіщень. У свою чергу, інтегрований веб-додаток React, який завантажується безпосередньо у внутрішньому контейнері WebView месенджера за адресою www.pralka.pp.ua, бере на себе завдання візуалізації складних графічних інтерфейсів, зокрема інтерактивної сітки часових слотів і матриці доступності обладнання. Такий розподіл обов'язків на клієнтському

рівні дозволяє розвантажити інтерфейс бота від складних текстових меню та забезпечити високу швидкість реакції інтерфейсу на дії користувача.

Серверний рівень системи (Backend layer) є логічним ядром програмного комплексу і базується на високонадійному веб-фреймворку Django (Django REST Framework) [12]. Серверна частина розгорнута на захищених обчислювальних потужностях та повністю ізольована від прямого доступу кінцевих користувачів. Бекенд виконує функцію централізованого постачальника даних (API Provider) і координує всі ключові бізнес-процеси: обробку реляційних запитів до бази даних PostgreSQL, валідацію вхідної інформації на предмет використання заборонених символів латиниці, автоматичний пошук дублікатів профілів, розрахунок динамічних добових і тижневих лімітів бронювань, а також обробку вхідних транзакцій та токенів карткових платежів від зовнішнього фінансового шлюзу LiqPay [13]. Взаємодія між клієнтським веб-додатком React та серверною частиною Django реалізована за допомогою асинхронних HTTP-запитів через спеціалізовану бібліотеку Axios за архітектурним стилем REST API.



Малюнок 3.1.1. Схема загальної архітектури інформаційної системи

Важливою особливістю проєктування архітектури є винесення тривалих у часі операцій та періодичних завдань в окремий асинхронний інфраструктурний шар, відокремлений від основного потоку обробки HTTP-запитів. Для цього в серверній архітектурі реалізовано чергу завдань на базі інструменту Celery, що взаємодіє з високопродуктивним брокером повідомлень RabbitMQ. Цей архітектурний підхід дозволяє перевести процеси масової таргетованої розсилки сповіщень, щохвилинні перевірки наближення часу сесій прання, автоматичні тригерні нагадування для неверифікованих акаунтів та операції ініціювання щомісячних рекурентних списань коштів у фоновий режим. Як

наслідок, основний API-сервер залишається повністю розблокованим і здатний миттєво приймати та обробляти нові запити від користувачів навіть під час виконання масових обчислювальних або мережевих операцій, що гарантує високу стабільність системи в моменти пікових навантажень у студентському гуртожитку.

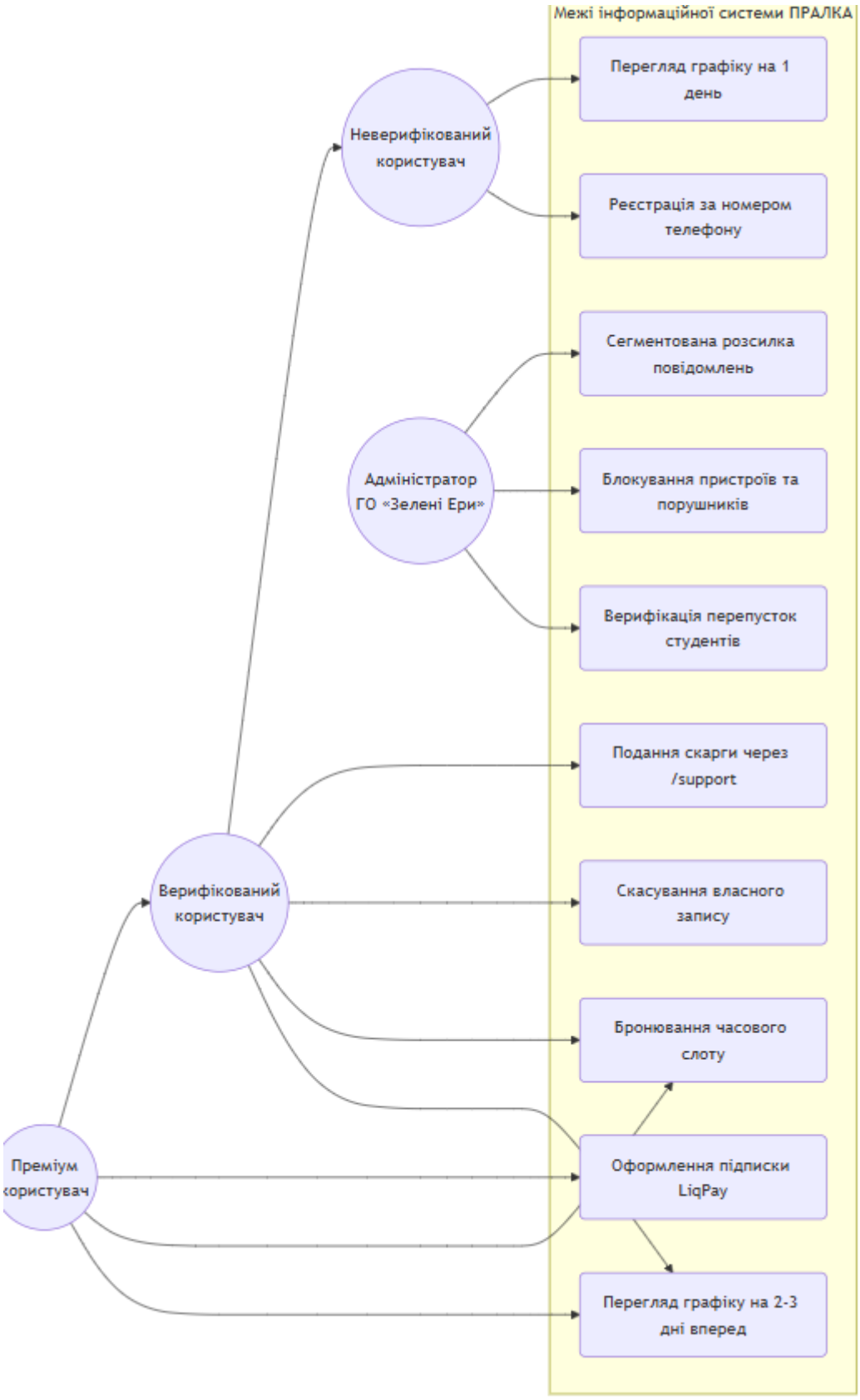
Безпека архітектурної взаємодії та захист конфіденційних даних забезпечуються наскрізним криптографічним шифруванням усього мережевого трафіку між клієнтами та сервером. Для цього на рівні зворотного проксі-сервера інтегровано автоматизовану систему генерації та оновлення SSL/TLS-сертифікатів Let's Encrypt, що дозволяє організувати роботу системи виключно за захищеними протоколами HTTPS та Secure WebSockets (WSS). Контейнеризація всіх архітектурних модулів (Django, Aiogram, PostgreSQL, Celery, RabbitMQ) за допомогою платформи Docker завершує формування цілісної клієнт-серверної моделі, забезпечуючи повну незалежність програмного забезпечення від конфігурації конкретного фізичного сервера та спрощуючи процес його подальшого розгортання та масштабування в інфраструктурі інших гуртожитків ПУЕТ та ПДАУ.

3.2. Графічне представлення архітектури та моделювання процесів за допомогою мови UML

Графічне моделювання за допомогою уніфікованої мови моделювання (UML) є стандартним та високоефективним інженерним підходом у галузі комп'ютерних наук, який дозволяє візуалізувати, специфікувати, конструювати та документувати артефакти складних програмних систем. Для інформаційного комплексу «Пралка» об'єктно-орієнтоване моделювання процесів [14] забезпечує чітке розуміння взаємодії між різними компонентами розподіленої інфраструктури, користувачами та зовнішніми сервісами ще до етапу безпосереднього написання програмного коду. У межах цього дослідження архітектурні та поведінкові особливості системи формалізуються за допомогою

трьох базових типів діаграм: діаграми варіантів використання, діаграми послідовності та діаграми діяльності.

Формування функціональних меж системи та визначення ролей взаємодії починається з побудови діаграми варіантів використання (Use Case Diagram), яка описує зв'язки між дійовими особами (акторами) та основними прецедентами. Головними акторами системи виступають мешканці гуртожитку, диференційовані за рівнями доступу, та адміністратор від громадської організації «Зелені Ери». Діаграма графічно відображає (див. малюнок 3.2.1), що неверифікований користувач має доступ лише до обмеженого переліку прецедентів, таких як первинна реєстрація та перегляд графіку на поточну добу. Верифікований акаунт розширює взаємодію на варіанти використання щодо бронювання часових комірок, скасування власних записів та надсилання сервісних скарг, тоді як преміум-користувач додатково зв'язується з прецедентом автоматичного відстеження вільних місць. Актор адміністратора наділений монопольними правами на прецеденти верифікації профілів, блокування несправних пральних машин, накладання лімітів-штрафів на порушників та ініціювання таргетованих розсилок сповіщень.

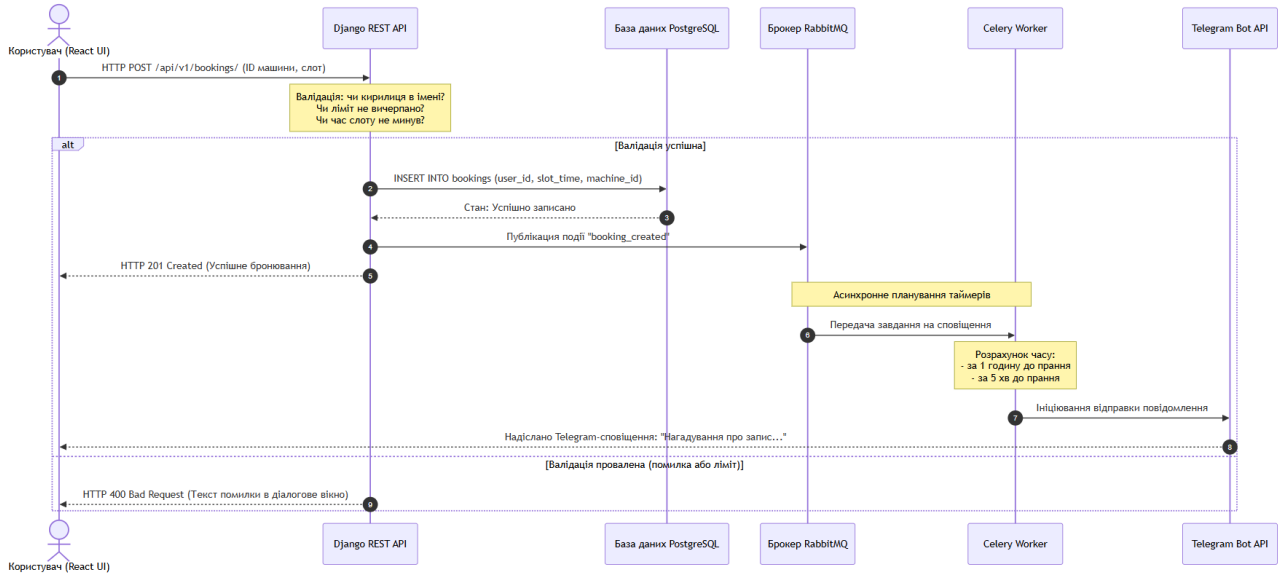


Малюнок 3.2.1. Діаграма «повноважень».

Для детального опису динамічної поведінки системи та часової послідовності обміну повідомленнями між розподіленими об'єктами проектується діаграма послідовності (Sequence Diagram). Вона фокусується на ключовому бізнес-процесі — бронюванні часового слоту через React Web-App

та подальшому плануванні черги сповіщень. На діаграмі вертикальними лініями життя (Lifelines) представляються такі компоненти, як клієнтський інтерфейс React, асинхронне ядро Django API, реляційна база даних PostgreSQL, брокер повідомлень RabbitMQ та фоновий обробник завдань Celery. Модель демонструє, як ініційований користувачем HTTP-запит передається через Axios до Django, де проходить валідацію на відповідність поточним лімітам та перевірку того, що обраний час слоту ще не розпочався. Після успішного запису транзакції в PostgreSQL, Django генерує подію для RabbitMQ, яку миттєво перехоплює Celery для планування асинхронних таймерів нагадувань у Telegram-боті за 1 годину та 5 хвилин до початку прання, що ілюструє високу реактивність та асинхронність розробленої архітектури.

Внутрішня логіка виконання операцій, алгоритми ухвалення рішень та розгалуження потоків управління в інформаційній системі деталізуються за допомогою діаграми діяльності (Activity Diagram), яка наведена на малюнку 3.2.2. Цей тип моделювання є критично важливим для візуалізації процесу реєстрації та багатоетапної валідації користувацьких даних. Діяльність системи починається з моменту введення студентом ПІБ у Telegram-боті. Потік управління спрямовується до умовного вузла рішення (Decision Node), який перевіряє набір символів: у разі виявлення латиниці діяльність переходить у стан виведення помилки, а за умови успішного використання кирилиці потік розділяється на паралельні гілки. Одна гілка забезпечує автоматичний пошук дублікатів за прізвищем у базі даних PostgreSQL, а інша — надсилає інтерактивну картку модерації з кнопками «Ок» та «Видалити акаунт» у закриту робочу групу адміністраторів. Окремий замкнений цикл на діаграмі діяльності описує роботу Celery Beat, який щотижня автоматично перевіряє прапорець верифікації в профілі та, за його відсутності, направляє користувача на тригерну гілку нагадувань про обмеження функціоналу.



Малюнок 3.2.2. Внутрішня логіка виконання операцій

У сукупності розроблені UML-моделі формують цілісний теоретичний каркас інформаційної системи «Пралка». Вони дозволяють суворо зафіксувати всі логічні обмеження та бізнес-вимоги — від приховування акційної кнопки підписки LiqPay після першої успішної транзакції до автоматичного фарбування стовпчиків зламаних пральних машин у червоний колір у Web-App. Графічне представлення процесів забезпечує високу якість проектування, мінімізує логічні помилки на етапі реалізації та слугує наочним архітектурним посібником.

3.3. Обґрунтування вибору технологічного стеку та інструментальних засобів реалізації

Ефективність, надійність та швидкість функціонування будь-якої розподіленої інформаційної системи безпосередньо залежать від обґрунтованого вибору технологічного стеку на кожному з її рівнів. Під час проєктування архітектури програмного комплексу «Пралка» ключовими критеріями відбору інструментальних засобів реалізації були забезпечення високої асинхронності для обробки потоків повідомлень, наявність вбудованих засобів безпеки персональних даних, швидкість розробки та можливість контейнеризації системи для її подальшого масштабування.

На клієнтському рівні для побудови інтерфейсу взаємодії з користувачами було обрано мову програмування Python [15] у поєднанні з фреймворком Aiogram. Вибір Aiogram обґрунтований його повністю асинхронною структурою, що базується на бібліотеці asyncio. Це дозволяє Telegram-боту обробляти тисячі сервісних команд та запитів одночасно без блокування потоку виконання, що є критично важливим для забезпечення швидкої відповіді сервісу в умовах щільної студентської спільноти. Для реалізації динамічної графічної матриці бронювання слотів застосовано мову JavaScript та фронтенд-бібліотеку React [16]. Використання концепції Single Page Application на базі React дозволило створити реактивний інтерфейс всередині вікна месенджера Telegram Web-App. Завдяки віртуальному дереву DOM оновлення стану пральних машин та часових комірок відбувається миттєво без перезавантаження веб-сторінки, що суттєво підвищує якість користувацького досвіду. Для забезпечення надійної асинхронної HTTP-комунікації між React Web-App та серверною частиною інтегровано спеціалізовану клієнтську бібліотеку Axios, яка надає гнучкі інструменти для обробки REST API запитів та автоматичної трансформації даних у формат JSON.

Ядром серверної архітектури системи виступає високорівневий веб-фреймворк Django (зокрема, розширення Django REST Framework) на базі мови програмування Python. Вибір Django обумовлений його монолітною надійністю, наявністю вбудованої об'єктно-реляційної моделі відображення даних (ORM), високим рівнем безпеки від типових вразливостей та готовим інструментарієм для побудови панелі адміністратора [17]. Django REST Framework дозволяє оперативно проектувати захищені ендпоінти для взаємодії з фронтендом та зовнішніми фінансовими сервісами. Як реляційну систему управління базами даних обрано PostgreSQL. Ця СУБД має високу репутацію в індустрії завдяки строгому дотриманню принципів ACID, підтримці складних транзакцій та здатності забезпечувати цілісність даних профілів, токенів підписок LiqPay та журналів записів студентів на прання.

Для оптимізації архітектурного навантаження та винесення важких операцій у фоновий режим було впроваджено зв'язку брокера повідомлень

RabbitMQ та черги асинхронних завдань Celery. RabbitMQ забезпечує високопродуктивне та надійне маршрутизування повідомлень-завдань, що генеруються основним API-сервером. Celery, функціонуючи у якості розподіленого обробника черг, бере на себе виконання всіх процесів, пов'язаних із тайм-менеджментом проекту: щохвилинне відстеження наближення часу прання, ініціювання рекурентних платежів на статутну діяльність громадської організації, масові таргетовані розсилки сповіщень та тригерні нагадування для неверифікованих користувачів через Celery Beat. Такий підхід гарантує, що основний веб-сервер Django залишається повністю вільним для миттєвого приймання вхідних HTTP-запитів від студентів.

Інфраструктурний рівень та розгортання системи повністю базуються на технології контейнеризації Docker. Використання Docker дозволяє ізолювати кожен сервіс — Django, Aiogram, PostgreSQL, Celery та RabbitMQ — в окремому контейнері, фіксуючи версії залежностей та операційного середовища. Це нівелює проблему розбіжностей конфігурацій між сервером розробки та Production-сервером, а також дозволяє безперешкодно та оперативно розгортати систему в інфраструктурі інших гуртожитків. Безпеку передачі даних та фінансових транзакцій забезпечує зворотний проксі-сервер Nginx [18] [19], інтегрований із центром сертифікації Let's Encrypt. Автоматичне наскрізне шифрування трафіку за протоколами HTTPS та WSS за допомогою безкоштовних криптографічних SSL-сертифікатів є обов'язковою інженерною умовою для захисту персональних даних мешканців та інтеграції з API LiqPay, Apple Pay та Google Pay.

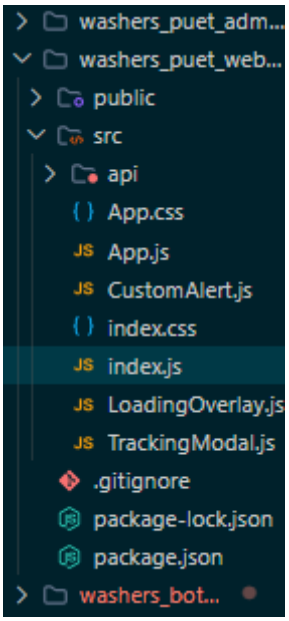
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

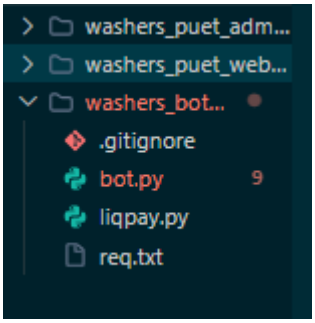
4.1. Програмна реалізація серверної логіки Django API та асинхронного Telegram-бота на базі Aiogram

Процес практичної реалізації інформаційної системи «Пралка» розпочався з розгортання та конфігурування серверної архітектури на базі високорівневого веб-фреймворку Django та інструментарію Django REST Framework (DRF). Серверна логіка спроєктована за принципом архітектурного стилю REST API, що дозволило відокремити обчислювальні процеси від інтерфейсу користувача та забезпечити єдину точку доступу до бази даних для Telegram-бота та клієнтського веб-додотка. На рівні об'єктно-реляційного відображення Django ORM було створено базові моделі даних, що описують сутності користувача, пральних машин, часових слотів та логів транзакцій. Для збереження інформації налаштовано реляційну СУБД PostgreSQL, причому конфігурація підключення винесена у захищені змінні оточення для відповідності вимогам безпеки. Всі сервери баз даних фізично розміщені в межах Європейського Союзу, що гарантує дотримання [20] європейських та вітчизняних стандартів захисту персональних даних мешканців гуртожитку. Структура компонентів системи Пралка наведена на малюнках 4.1.1-4.1.3

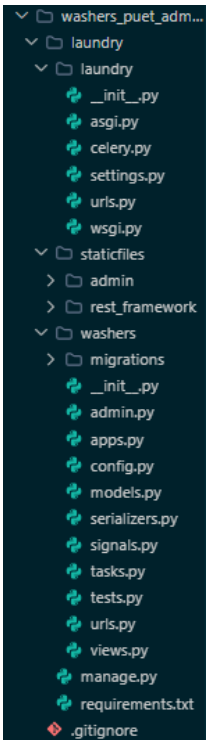
Наступним етапом розробки стало створення асинхронного чат-бота з використанням спеціалізованої бібліотеки Aiogram версії 3.x на базі мови програмування Python. Вибір асинхронної парадигми дозволив ефективно вирішити задачу високої конкурентності, коли сотні студентів одночасно взаємодіють з інтерфейсом у пікові години. В архітектурі бота реалізовано систему диспетчеризації запитів (Dispatcher) та модулі кінцевих автоматів (Finite State Machine) для покрокового збору інформації під час первинної реєстрації. Взаємодія бота із сервером Django відбувається через внутрішні захищені HTTP-запити, що унеможливлює прямий доступ клієнтського коду до бази даних. В інтерфейсі бота запрограмовано обробку обов'язкових службових

команд, зокрема команди виклику технічної підтримки /support, яка виводить контактну інформацію громадської організації «Зелені Ери», та команди /subscribe_cancel для оперативного управління фінансовими підписками

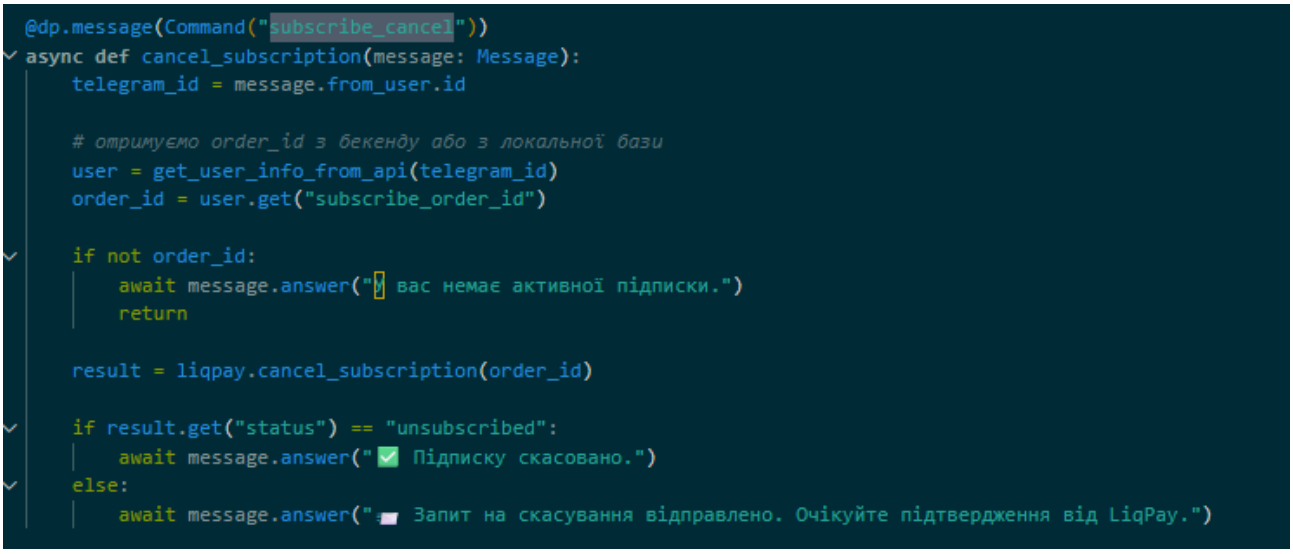




мешканців (див. малюнок 4.1.4)



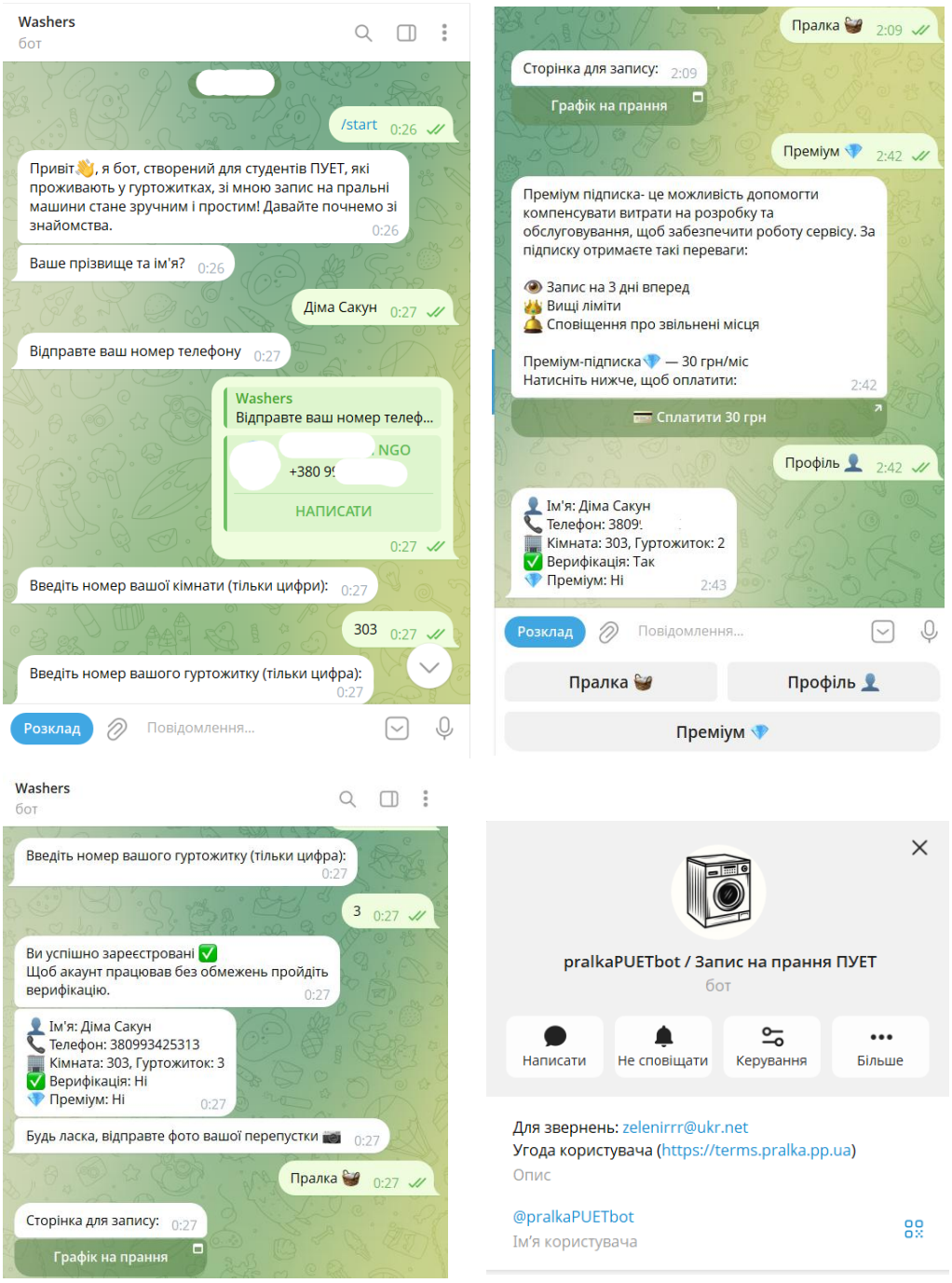
Малюнки 4.1.1.- 4.1.3. Структура компонентів системи «Пралка»



```python
@dp.message(Command("subscribe_cancel"))
async def cancel_subscription(message: Message):
    telegram_id = message.from_user.id

    # отримуємо order_id з бекенду або з локальної бази
    user = get_user_info_from_api(telegram_id)
    order_id = user.get("subscribe_order_id")

    if not order_id:
        await message.answer("У вас немає активної підписки.")
        return

    result = liqpay.cancel_subscription(order_id)

    if result.get("status") == "unsubscribed":
        await message.answer("✅ Підписку скасовано.")
    else:
        await message.answer("📩 Запит на скасування відправлено. Очікуйте підтвердження від LiqPay.")
```

Малюнок 4.1.4. Реалізація команди / subscribe_cancel

Особливу увагу при програмній реалізації серверного ядра було приділено модулям аутентифікації та валідації вхідних даних користувачів. Під час первинного старту бота система впроваджує двофакторну перевірку: зчитується унікальний ідентифікатор Telegram ID та здійснюється обов'язковий запит на надсилання верифікованого номеру телефону через вбудований інтерфейс месенджера. Для запобігання зловживанням, хаотичній реєстрації фейкових акаунтів та спробам обходу лімітів, на рівні Django-серіалізаторів розроблено програмний фільтр на основі регулярних виразів. Цей фільтр повністю блокує спроби використання латинських символів під час введення імені та прізвища користувача, дозволяючи збереження профілю лише за умови використання кириличного набору літер. Одночасно з цим у моделі профілю реалізовано автоматичне присвоєння початкового невизначеного статусу доступу, який накладає жорсткі обмеження на використання побутових ресурсів.



Малюнки 4.1.5-4.1.8. Вигляд телеграм боту «Пралка»

Для забезпечення ергономічної взаємодії користувача з графічною матрицею бронювання, в архітектуру Telegram-бота інтегровано модуль генерації інлайнових клавіатур (InlineKeyboardMarkup) зі спеціальними кнопками типу WebApp. Програмний код бота динамічно формує посилання на дзеркало веб-додотка www.pralka.pp.ua, додаючи до запиту унікальну криптографічну сигнатуру (initData), що генерується серверами Telegram. Під час переходу студента за кнопкою запису, серверна частина Django розшифровує цей токен за допомогою секретного ключа бота, автоматично підтверджуючи автентичність сесії користувача без необхідності введення

додаткових паролів. Таким чином, успішна програмна інтеграція Django API та Aiogram дозволила створити стабільний, захищений та швидкий серверний фундамент інформаційної системи, готовий до обробки великих масивів асинхронних побутових запитів.

4.2. Розробка динамічного клієнтського інтерфейсу React Web-App та інтеграція платіжної системи LiqPay

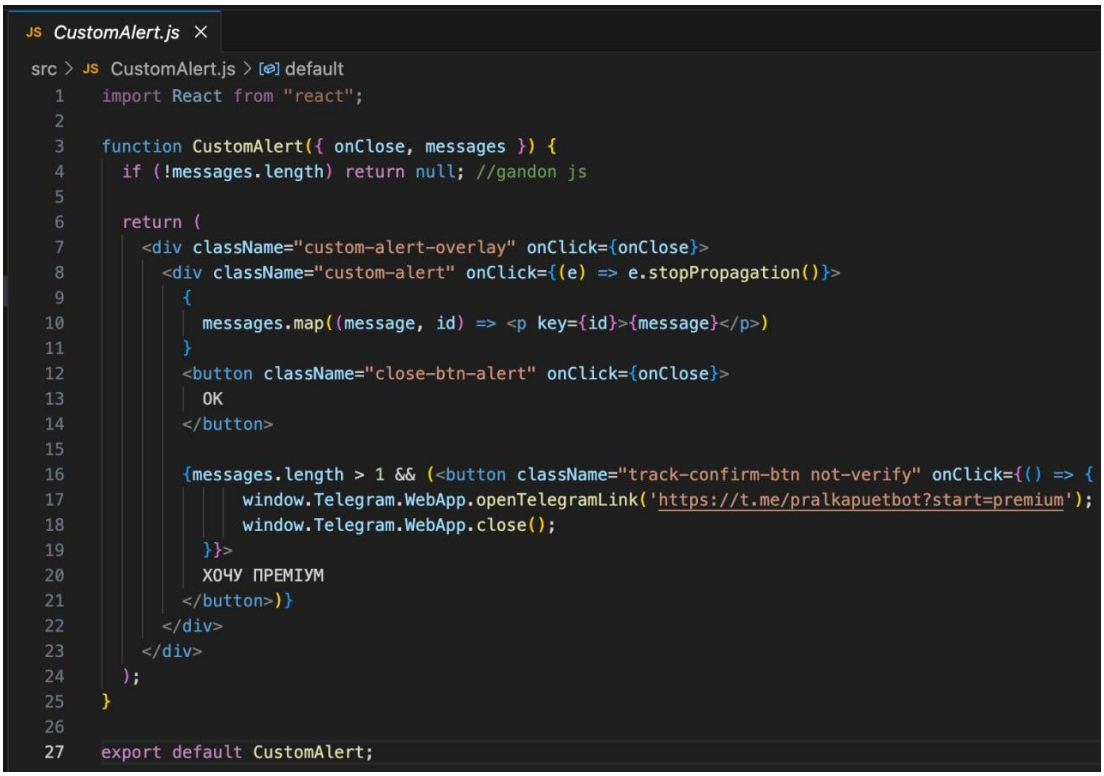
Розробка клієнтської частини інформаційної системи у вигляді веб-додатка React Web-App дозволила створити інтерактивне табло високого рівня ергономіки. Односторінковий додаток розгорнуто за веб-адресою www.pralka.pp.ua та оптимізовано для безперешкодного завантаження всередині месенджера Telegram за технологією WebApp. Інтерфейс додатка побудовано за компонентним принципом, де основним елементом є динамічна сітка часових слотів. Кожен стовпчик цієї матриці представляє окрему фізичну пральну машину, а рядки — дискретні годинні інтервали. Для забезпечення реактивності інтерфейсу використано хуки стану React, що дозволяє миттєво оновлювати стан доступності часових комірок після успішного запису користувача. Важливою інженерною вимогою при розробці дизайну інтерфейсу стала реалізація механізму інфраструктурного блокування: у разі технічної несправності пристрою адміністратор встановлює часовий діапазон блокування, після чого відповідний стовпчик у Web-App повністю забарвлюється у червоний колір, оскільки жовте маркування не відповідав загальному стилістичному оформленню додатка. Це технічне рішення повністю блокує можливість кліків студентів на будь-які слоти цієї машини. Крім того, на рівні інтерфейсу впроваджено логічне обмеження, яке унеможливлює вибір часового слоту, початок якого за поточним часом сервера вже настав, що мінімізує помилкові натискання наприкінці години та не дозволяє створювати записи, які вже неможливо скасувати. На малюнку 4.2.1 наведено фрагмент коду, який відповідає за асинхронне бронювання пральних машин.

```
168
169     const updateTrackingForDate = async (range) => {
170       try {
171         const updatedTrackedDates = [...(user.tracked_dates || [])];
172         const existingIndex = updatedTrackedDates.findIndex(t => t.startsWith(date));
173         if (existingIndex !== -1) {
174           updatedTrackedDates.splice(existingIndex, 1);
175         }
176         if (range) {
177           updatedTrackedDates.push(`${date} ${range}`);
178         }
179
180         await updateUser(telegram_id, { tracked_dates: updatedTrackedDates });
181         setUser((prev) => ({ ...prev, tracked_dates: updatedTrackedDates }));
182         setTrackingActive(!!range);
183         setTrackedRangeForDate(range ? `${date} ${range}` : null);
184       } catch (e) {
185         console.error("Не вдалося оновити tracked_dates", e);
186       }
187     };
188
189     const handleBook = async (washerId, hour) => {
190       setIsLoading(true); // Початок завантаження дії
191       try {
192         await bookSlot({
193           washer: washerId,
194           telegram_id: telegram_id,
195           date,
196           start_time_input: hour,
197         });
198         const updatedSlots = await getSlots(washerId, date);
199         setSlotsMap((prev) => ({ ...prev, [washerId]: updatedSlots }));
200       } catch (err) {
201         let message = "Комірка вже заброньована або сталась інша помилка(";
202         if (err.response && err.response.data && typeof err.response.data === "object") {
203           const errors = err.response.data;
204           if (errors.non_field_errors?.[0]) {
205             message = errors.non_field_errors[0];
206           } else if (Array.isArray(errors) && errors.length > 0) {
207             message = errors[0];
208           }
209         }
210         setAlertMessage([message]);
211       } finally {
```

Малюнок 4.2.1. Реалізація асинхронного бронювання пральних машин

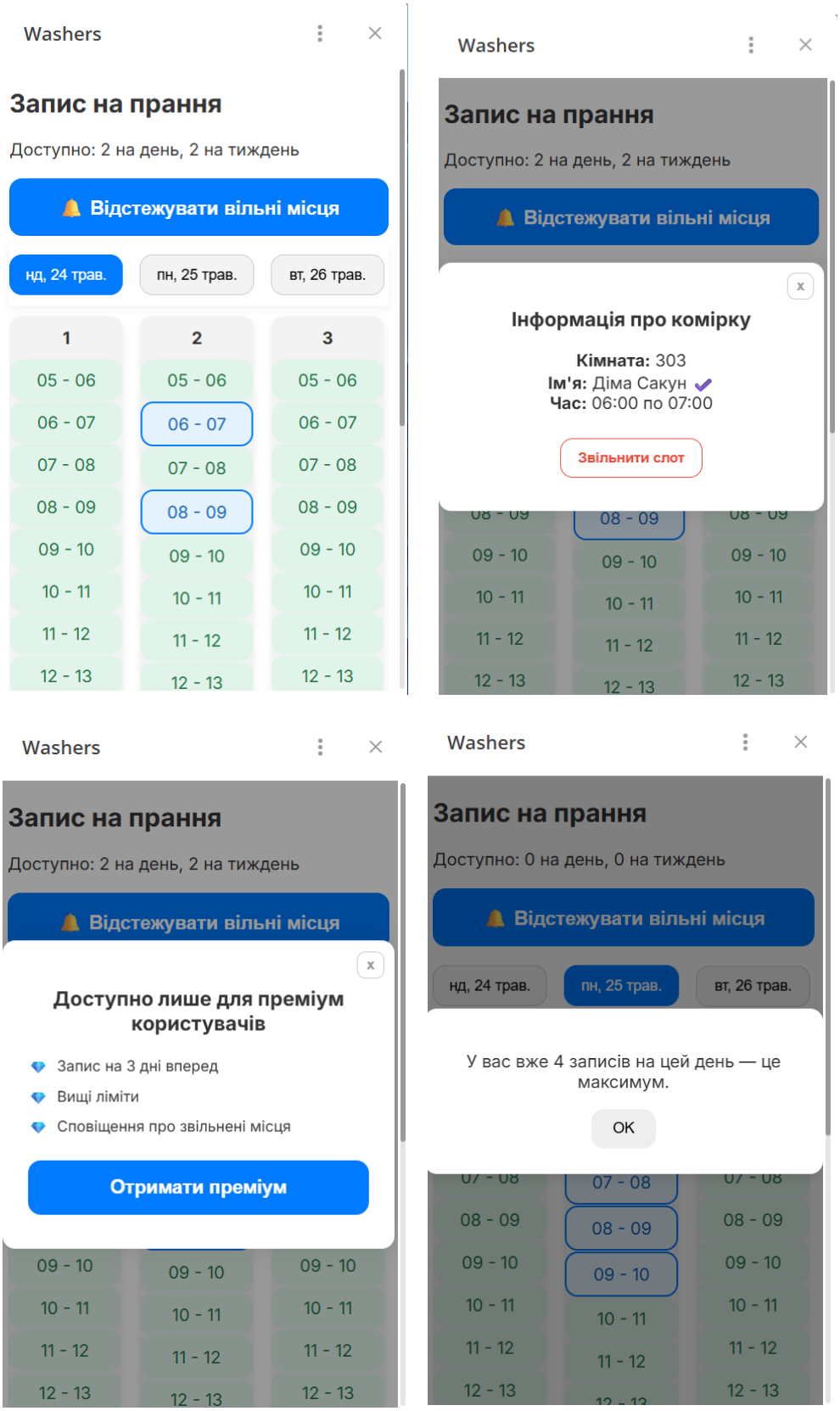
Програмна логіка фронтенд-частини тісно інтегрована з рольовою моделлю доступу Django API. Для верифікованих користувачів, які підтвердили свій статус проживання, у діалоговому вікні інформації про поточний слот поруч із їхнім іменем відображається спеціальна синя позначка верифікації, що додає впевненості у справжності акаунту для всієї студентської спільноти. Натомість для неверифікованих мешканців, які мають жорсткі обмеження лімітів, у Web-App реалізовано постійне демонстраційне діалогове вікно з

попереджувальним текстом про те, що профіль не пройшов верифікацію, функціонал обмежено, а для зняття лімітів необхідно надіслати перепустку або оформити підписку. У цьому ж вікні інтегровано активну кнопку «Хочу преміум». Код відповідної функції наведено на малюнку 4.2.2.



Малюнок 4.2.2. Код банеру преміум підписки

Окремо було модернізовано розділ підтримки користувачів. Додано функціональну кнопку «Зв'язатися з адміністратором», при натисканні на яку виводиться сервісний текст із вказівкою надсилати свої запитання, пропозиції для покращення сервісу та скарги на електронну пошту громадської організації zelenirrr@ukr.net. Також у межах покращення інтерфейсу та уникнення міжособистісних конфліктів кнопку затримки прання було перейменовано на «скарга на затримку прання іншим користувачем», що чітко специфікує призначення цієї функції для студентів. Для користувачів без верифікації спроби використання цих функцій блокуються з виведенням повідомлення про потребу верифікувати профіль.



Малюнки 4.2.3-4.2.6. Вигляд веб додатку системи «Пралка»

Фінансовата стійкість та самоокупність проєкту забезпечується успішною інтеграцією платіжної системи LiqPay для збору внесків на статутну діяльність громадської організації «Зелені Ери». Програмна реалізація модуля монетизації передбачає відображення детального тексту переваг Преміум-підписки перед

блоком оплати, який інформує про можливість запису на три дні вперед, вищі ліміти та автоматичне сповіщення про місця, що звільнилися. Вартість стандартної підписки становить тридцять гривень на місяць і списується за рекурентним принципом. Для залучення користувачів на початковому етапі впроваджено механізм акційної ціни на перший місяць використання, яка становить одну гривню. З точки зору програмної логіки, після успішного оформлення підписки та проведення першої транзакції акційна кнопка автоматично приховується з інтерфейсу Web-App, а користувач переводиться на регулярний тариф. Під час проведення транзакцій через безконтактні методи Apple Pay та Google Pay, система реалізує додатковий рівень безпеки та перевірки даних: серверна частина здійснює запит до платіжного шлюзу для отримання реального прізвища, імені та по батькові власника картки, що дозволяє адміністраторам автоматично зіставити ці відомості з даними, вказаними при реєстрації у Telegram-профілі, запобігаючи використанню чужих або вигаданих персональних даних.

4.3. Алгоритмічна реалізація модулів валідації, захисту від дублікатів та фонових завдань Celery

Програмна реалізація алгоритмічного шару інформаційної системи «Пралка» зосереджена на забезпеченні цілісності бази даних, автоматизації контролю користувацьких дій та оптимізації фонових процесів. Одним із ключових модулів цієї підсистеми є алгоритм виявлення та запобігання створенню дублікатів облікових записів. Під час надсилання користувачем заявки на верифікацію профілю (завантаження фотокопії перепустки) у системі запускається тригерна функція на базі Django ORM. Вона здійснює автоматичний пошук у таблиці бази даних PostgreSQL на предмет збігу прізвища з уже існуючими записами. У разі виявлення потенційного дубліката система не блокує дію автоматично, а маркує заявку спеціальним прапорцем підозри та надсилає відповідне сервісне повідомлення у закриту Telegram-групу для адміністраторів із чітким зазначенням знайденого збігу. Для зручності

диспетчеризації в адміністративній панелі Django розроблено спеціалізоване текстове поле пошуку та фільтрації, яке дозволяє модераторам оперативно вивести повний список користувачів з однаковими або схожими анкетними даними для ухвалення остаточного рішення.

Паралельно з перевіркою дублікатів на рівні серіалізаторів Django REST Framework функціонує алгоритм жорсткої валідації текстових полів профілю. Програмний код реалізує регулярні вирази, які повністю забороняють використання латинських символів, цифрових значень або спецсимволів під час заповнення полів імені та прізвища, дозволяючи збереження лише кириличного набору літер. Це унеможливлює хаотичне внесення вигаданих даних і спрощує процес зіставлення профілів із реальними списками мешканців гуртожитку. Іншим критично важливим алгоритмом валідації реального часу є контроль часових міток при взаємодії з графіком бронювання у React Web-App. Коли користувач намагається створити новий запис або скасувати існуючий, серверна логіка Django порівнює поточний системний час із часом початку обраного слоту. Якщо час початку комірки вже настав або минув, запит відхиляється з поверненням HTTP-помилки, що захищає систему від випадкових помилкових бронювань наприкінці години та дисциплінує студентів, не дозволяючи скасовувати записи після початку сесії прання.

Важливим елементом підтримання дисципліни та автоматизації тайм-менеджменту в системі є архітектурний модуль фонових завдань, реалізований за допомогою Celery та брокера повідомлень RabbitMQ. Для координації дій мешканців гуртожитку №2 ПУЕТ було розроблено серію асинхронних завдань, які виконуються за розкладом. Програмний модуль Celery щохвилини сканує таблицю активних бронювань і, виходячи з розрахункових часових міток, ініціює відправку тригерних повідомлень через Telegram Bot API. Сервіс автоматично надсилає студенту нагадування про заплановане прання рівно за одну годину, а потім повторно за п'ять хвилин до початку обраного часового слоту. Окрім цього, для оптимізації пропускної здатності пральних машин та ліквідації затримок інфраструктури, за п'ять хвилин до завершення відведеного часу броні Celery генерує та надсилає користувачу повідомлення з проханням

завершити процес, вивантажити особисті речі та звільнити пристрій для наступного за списком мешканця.

Окрему логічну гілку в структурі фонових обробників Celery Beat займає алгоритм роботи з неверифікованими акаунтами. Якщо користувач пройшов первинну реєстрацію за номером телефону, але не завантажив перепустку до гуртожитку для підтвердження особи, Celery Beat автоматично запускає тригерне завдання рівно на наступний день після реєстрації. Користувач отримує офіційне сповіщення про те, що функціонал сервісу для нього обмежено, а для зняття лімітів необхідно надіслати перепустку або оформити преміум-підписку, під час оплати якої адміністрація зможе перевірити справжнє ім'я платника. Якщо користувач ігнорує це повідомлення, алгоритм повторює розсилку кожні сім днів до моменту зміни статусу профілю. Взаємодія з модераторами під час реєстрації нових користувачів також автоматизована на рівні бота для уникання створення фальшивих акаунтів. Дані надходять у робочу групу у вигляді картки з двома інтерактивними кнопками: натискання кнопки «Ок» просто прибирає блоки вибору з інтерфейсу повідомлення, тоді як натискання кнопки «Видалити акаунт» повністю очищує запис у базі даних та надає адміністратору можливість надіслати користувачу сповіщення із чітким зазначенням причини видалення.

4.4. Фінальне тестування, аналіз надійності та результати експериментального впровадження системи

Фінальний етап розробки інформаційної системи «Пралка» передбачав проведення комплексного багаторівневого тестування всіх функціональних блоків та оцінку показників її надійності в умовах реального навантаження. Процес верифікації програмного коду охоплював функціональне тестування інтерфейсу Telegram-бота, реактивної матриці слотів React Web-App, а також перевірку коректності спрацьовування асинхронних черг Celery. Особливу увагу під час тестування було приділено модулю монетизації та обробки регулярних платежів. На першому етапі було виконано серію тестових транзакцій на

рахунок громадської організації «Зелені Ери» з використанням пісочниці та тестових ключів платіжного шлюзу LiqPay, що дозволило перевірити логіку автоматичного приховування акційної кнопки після першої оплати та механізм скасування підписки. Після успішного завершення емуляції платіжних процесів до системи було підключено бойові ключі громадської організації, за допомогою яких проведено фінальні перевірки реального списання коштів, у тому числі з верифікацією передачі анкетних даних через Apple Pay та Google Pay.

Аналіз надійності архітектурного стеку технологій, розгорнутого за допомогою контейнеризації Docker під захистом сертифікатів Let's Encrypt, підтвердив високу стійкість системи до відмов. Протягом усього періоду експериментального впровадження та підконтрольної експлуатації інформаційного комплексу не було зафіксовано жодного критичного технічного збою чи колізії при паралельному бронюванні слотів різними користувачами. Стабільність роботи серверної логіки Django API в комбінації з брокером повідомлень RabbitMQ дозволила забезпечити безперебійне функціонування сервісу в режимі двадцять чотири на сім. Асинхронна архітектура бота на базі Aiogram повністю нівелювала затримки при обробці вхідних пакетів даних у години пікового навантаження, коли мешканці гуртожитку одночасно відкривали графік для планування побутових процесів на наступні дні.

Результати експериментального впровадження системи на базі студентського гуртожитку номер два Полтавського університету економіки і торгівлі засвідчили високу соціальну та організаційну ефективність розробленого програмного продукту. За час дослідної експлуатації в системі успішно зареєструвалися 267 активних користувачів із числа мешканців, а сумарна кількість оброблених та зафіксованих у базі даних PostgreSQL бронювань часових слотів склала 1982 операції. Згідно з результатами проведеного соціологічного моніторингу та анкетування користувачів, понад 82 відсотки опитаних студентів офіційно підтвердили ліквідацію міжособистісних конфліктів, пов'язаних із черговістю прання, та відзначили суттєве підвищення загального рівня щоденного побутового комфорту. Успішна апробація сервісу та

високі експлуатаційні показники дозволили сформувати стратегічні плани щодо подальшого масштабування проекту, які включають розгортання інформаційної системи у гуртожитках номер один та чотири Полтавського університету економіки і торгівлі, а також впровадження платформи в інфраструктуру гуртожитків Полтавської державної аграрної академії.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну науково-практичну задачу з автоматизації спільного використання обмежених побутових ресурсів у студентському середовищі через проєктування, програмну реалізацію та розгортання розподіленої інформаційної системи «Пралка». Впровадження цієї системи дозволило замінити застарілі та неефективні паперові журнали або хаотичні живі черги впорядкованим віртуальним розкладом, що забезпечив справедливий і прозорий розподіл часових ресурсів, а також суттєво знизив навантаження на локальні електромережі гуртожитків закладу вищої освіти в умовах сучасного дефіциту енергопотужностей в Україні.

На основі аналізу існуючих світових та вітчизняних аналогів обґрунтовано доцільність розробки індивідуального програмного комплексу, повністю незалежного від брендів пральних машин та дороговартісних апаратних контролерів інтернету речей. Про проєктування архітектури за клієнт-серверною моделлю дозволило об'єднати асинхронний Telegram-бот на базі фреймворку Aiogram як первинну точку входу та реактивний веб-додаток React Web-App для візуалізації сітки часових слотів. Серверний рівень на базі веб-фреймворку Django у поєднанні з реляційною СУБД PostgreSQL та фоновими обробниками завдань Celery й RabbitMQ забезпечив високу стійкість, надійність обробки великих потоків асинхронних запитів і безпеку персональних даних мешканців.

Практична реалізація системи дозволила успішно впровадити строгий алгоритмічний контроль внутрішніх бізнес-правил, включаючи автоматичне відхилення записів на слоти, що вже розпочалися за часом сервера, заборону використання латинських символів у полях ідентифікації та механізми виявлення дублікатів облікових записів мешканців. Інтеграція платіжного шлюзу LiqPay для рекурентних мікроплатежів забезпечила фінансову самоокупність проєкту на користь громадської організації «Зелені Ери» за рахунок надання розширеного Преміум-функціоналу. Результати

експериментальної апробації в гуртожитку №2 Полтавського університету економіки і торгівлі, де залучено 267 активних користувачів та успішно оброблено 1982 бронювання при повній відсутності технічних збоїв, підтвердили високу ефективність сервісу. Понад 82% опитаних студентів офіційно засвідчили ліквідацію міжособистісних конфліктів та значне підвищення щоденного побутового комфорту.

Важливо! Наразі ведуться переговори з адміністрацією ПУЕТу щодо впровадження системи у всіх гуртожитках університету. Сподіваємося, на підтримку проєкту від нашої alma mater для покращення життя всіх студентів у гуртожитках та демонстрації реального впливу студентів на життєдіяльність ПУЕТу і студентоорієнтованість від адміністрації. Частина функціоналу бота доступна до перегляду на час презентації цієї роботи за іншими тимчасовими посиланнями, зокрема ім’я тимчасового бота: @hbHbLYIbg63bvKUYbot , а веб додаток: https://washers.ex1side.xyz/admin , Нормативна документація до проєкту продовжує знаходитися за веб адресою https://terms.pralka.pp.ua/ .

РЕКОМЕНДАЦІЇ

За результатами успішного впровадження та підконтрольної експлуатації інформаційного комплексу «Пралка» сформовано низку науково-практичних рекомендацій щодо його подальшого використання, розвитку та технічної модернізації. По-перше, зважаючи на високу стабільність серверної архітектури та позитивний соціально-економічний ефект, рекомендується масштабувати розроблений сервіс і здійснити його розгортання в інших побутових кімнатах студентського кампусу Полтавського університету економіки і торгівлі, зокрема в гуртожитках №1 та №4. Завдяки повній контейнеризації всіх модулів у середовищі Docker цей процес потребуватиме мінімальних часових та обчислювальних витрат із боку адміністрації ЗВО.

По-друге, перспективним напрямом розвитку проєкту є вихід за межі одного навчального закладу та адаптація платформи для впровадження в інфраструктуру гуртожитків інших закладів вищої освіти регіону, насамперед у Полтавському державному аграрному університеті (ПДАУ). Для цього доцільно модернізувати серверне ядро Django API з метою підтримки мультиорендної архітектури (Multi-tenancy), що дозволить надійно ізолювати дані різних університетів та гуртожитків у межах єдиної бази даних PostgreSQL, зберігаючи при цьому централізоване управління розсилками та фінансовими транзакціями громадської організації «Зелені Ери».

По-третє, у межах подальшої технічної еволюції програмного комплексу рекомендується розширити функціонал клієнтського React-інтерфейсу за рахунок інтеграції прогнозних моделей на основі аналізу історичних даних бронювань. Це дозволить автоматично підсвічувати для користувачів години найменшої завантаженості побутових кімнат, сприяючи ще більш рівномірному розподілу електронавантаження протягом доби. Також доцільно додати модуль розширеної аналітики в панель адміністратора для автоматичного формування звітів про технічний знос пральних машин на основі кількості відпрацьованих

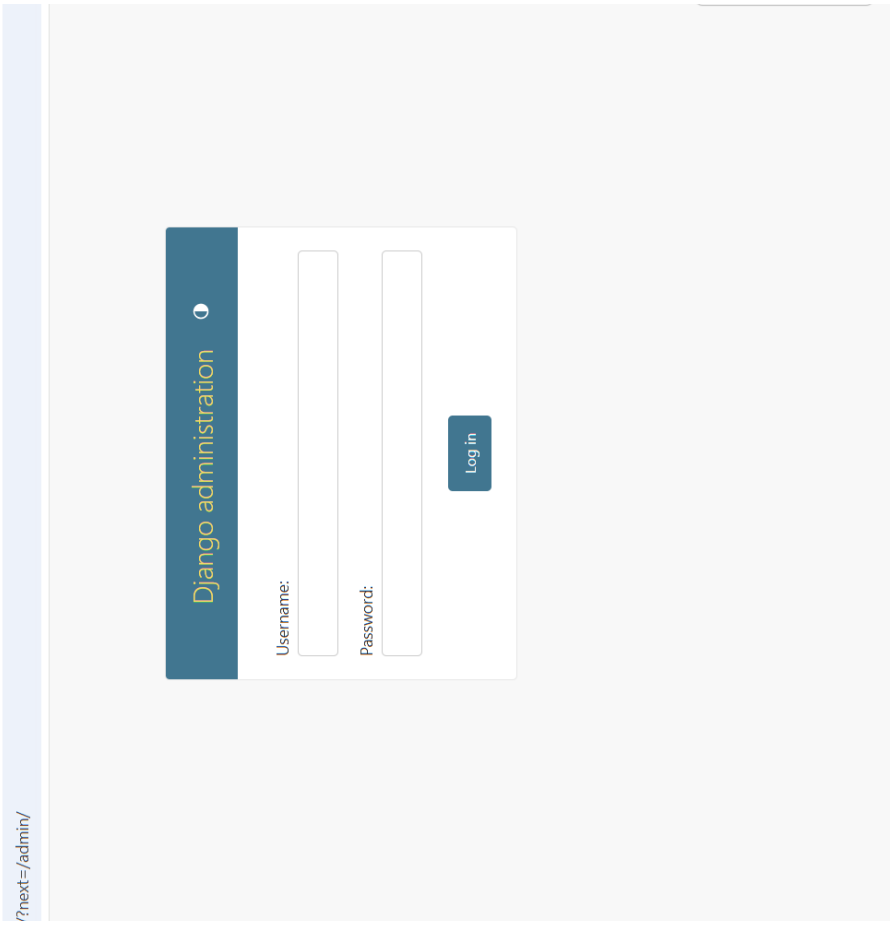
циклів, що дозволить заздалегідь планувати ремонтні роботи та мінімізувати терміни повного червоного блокування пристроїв у веб-додатку.

СПИСОК ЛІТЕРАТУРИ

1. Шаховська Н. Б., Нога Р. Ю. Об'єктно-орієнтоване моделювання програмних систем за допомогою мови UML. Львів : Видавництво Львівської політехніки, 2018. 184 с.
2. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ : Видавнича група BHV, 2006. 384 с.
3. Документація фреймворку Aiogram 3.x : офіційний сайт. URL: https://docs.aiogram.dev (дата звернення: 24.04.2026).
4. Banks A., Porcello E. Learning React: Modern Patterns on Developing Real-World Web Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 370 p.
5. Документація черги завдань Celery : офіційний сайт. URL: https://docs.celeryq.dev (дата звернення: 24.04.2026).
6. Специфікація брокера повідомлень RabbitMQ : офіційний сайт. URL: https://www.rabbitmq.com (дата звернення: 24.04.2026).
7. Mouat A. Using Docker: Developing and Deploying Software with Containers. Sebastopol : O'Reilly Media, 2015. 354 p.
8. Специфікація платіжного шлюзу LiqPay API : офіційний сайт. URL: https://www.liqpay.ua/documentation (дата звернення: 24.04.2026).
9. Ситник В. Ф. Проектування баз даних : навч. посіб. Київ : КНЕУ, 2001. 254 с.
10. Tanenbaum A. S., Bos H. Modern Operating Systems. 4th ed. Boston : Pearson, 2015. 1136 p.
11. Глибовець М. М., Олецький О. В. Архітектура систем програмного забезпечення. Київ : НаУКМА, 2006. 312 с.
12. Документація інструменту Django REST Framework : офіційний сайт. URL: https://www.django-rest-framework.org (дата звернення: 24.04.2026).
13. Офіційний сайт фінансового шлюзу LiqPay. URL: https://www.liqpay.ua (дата звернення: 24.04.2026).
14. Бублик В. В. Об'єктно-орієнтоване програмування : підручник. Київ : IT-книга, 2015. 424 с.

15. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1648 p.
16. Документація бібліотеки React.js : офіційний сайт. URL: https://react.dev (дата звернення: 24.04.2026).
17. Документація веб-фреймворку Django : офіційний сайт. URL: https://docs.djangoproject.com (дата звернення: 24.04.2026).
18. Nedelcu I. Nginx Essentials. Birmingham : Packt Publishing, 2015. 140 p.
19. Документація зворотного проксі-сервера Nginx : офіційний сайт. URL: https://nginx.org/en/docs/ (дата звернення: 24.04.2026).
20. Siriwardena P. Microservices Security in Action. New York : Manning Publications, 2020. 450 p.

ДОДАТОК А. Вікно входу в адміністративну панель



ДОДАТОК Б. Фрагменти коду телеграм боту

```
подтверждения
k_query(F.data.startswith("finishreg_"))
erify_request(callback: CallbackQuery, state: FSMContext):
back.data.split('_')[1] == 'verify':
it bot.send_message(callback.from_user.id, "Будь ласка, відправте фото вашої перепустки 📷")
it callback.answer("Очікуємо фото")
it state.set_state(ConfirmStates.waiting_for_photo)

it callback.answer('Почнемо реєстрацію наново.')
it cmd_register(message=callback.message, state=state)
bot.send_message(callback.from_user.id, str(callback.message))
ot.edit_message_text(chat_id=callback.from_user.id, message_id=callback.message.message_id, text=callback.message.text)

(ConfirmStates.waiting_for_photo, F.photo)
andle_photo(message: Message, state: FSMContext):
 message.photo[-1].file_id
ta = get_user_info_from_api(message.from_user.id)
 = f"📩 Фото перепустки користувача: {user_data['name']}, @{message.from_user.username if message.from_user.username != None else ""
lineKeyboardMarkup(inline_keyboard=

 [
    InlineKeyboardButton(text="✅ Підтвердити", callback_data=f"confirm_{message.from_user.id}"),
    InlineKeyboardButton(text="❌ Відмова", callback_data=f"reject_{message.from_user.id}")
 ]

ot.send_photo(CHAT_ID, photo, caption=caption, reply_markup=kb)
essage.reply("Фото відправлено на перевірку. ⏳")
tate.clear()
```

```
подтверждения
k_query(F.data.startswith("finishreg_"))
erify_request(callback: CallbackQuery, state: FSMContext):
back.data.split('_')[1] == 'verify':
it bot.send_message(callback.from_user.id, "Будь ласка, відправте фото вашої перепустки 📷")
it callback.answer("Очікуємо фото")
it state.set_state(ConfirmStates.waiting_for_photo)

it callback.answer('Почнемо реєстрацію наново.')
it cmd_register(message=callback.message, state=state)
bot.send_message(callback.from_user.id, str(callback.message))
ot.edit_message_text(chat_id=callback.from_user.id, message_id=callback.message.message_id, text=callback.message.text)

(ConfirmStates.waiting_for_photo, F.photo)
andle_photo(message: Message, state: FSMContext):
 message.photo[-1].file_id
ta = get_user_info_from_api(message.from_user.id)
 = f"📩 Фото перепустки користувача: {user_data['name']}, @{message.from_user.username if message.from_user.username != None else ""
lineKeyboardMarkup(inline_keyboard=

 [
    InlineKeyboardButton(text="✅ Підтвердити", callback_data=f"confirm_{message.from_user.id}"),
    InlineKeyboardButton(text="❌ Відмова", callback_data=f"reject_{message.from_user.id}")
 ]

ot.send_photo(CHAT_ID, photo, caption=caption, reply_markup=kb)
essage.reply("Фото відправлено на перевірку. ⏳")
tate.clear()
```

ДОДАТОК Б. Продовження

```
(RegistrationStates.waiting_for_phone_number)
et_phone(message: Message, state: FSMContext):
plyKeyboardMarkup(
board=[[KeyboardButton(text="Поділитись номером", request_contact=True)]],
ize_keyboard=True

essage.answer("Вручну написаний номер не приймається, номер потрібно відправити лише через кнопку:", reply_markup=kb)

(RegistrationStates.waiting_for_room_number)
et_room(message: Message, state: FSMContext):
message.text.isdigit():
urn await message.answer("Номер кімнати має складатись тільки з цифр!")
tate.update_data(room_number=message.text)
essage.answer("Введіть номер вашого гуртожитку (тільки цифра):")
tate.set_state(RegistrationStates.waiting_for_dormitory_number)

(RegistrationStates.waiting_for_dormitory_number)
et_dorm(message: Message, state: FSMContext):
message.text.isdigit():
urn await message.answer("Номер гуртожитку має бути тільки цифрою!")

ta = await state.get_data()

_exists(message.from_user.id):
ated = update_user(
 telegram_id=message.from_user.id,
 name=user_data['name'],
 telegram_username=message.from_user.full_name,
 telegram_user=message.from_user.username,
 phone_number=user_data['phone_number'],
 room_number=user_data['room_number'],
 dormitory_number=message.text
```

```
рация
md_register(message: Message, state: FSMContext):
essage.answer("Ваше прізвище та ім'я?", reply_markup=ReplyKeyboardRemove())
tate.set_state(RegistrationStates.waiting_for_name)

(RegistrationStates.waiting_for_name)
et_name(message: Message, state: FSMContext):
message.text.strip()

ація: тільки кирилиця (може містити пробіли, дефіс)
re.fullmatch(r"^[А-ЯІЇЄҐа-яіїєґ''\-\s]+$", name):
it message.answer("❌ Ім'я та прізвище мають бути українською мовою(тільки кирилиця). Спробуйте ще раз.")
urn

tate.update_data(name=name)

plyKeyboardMarkup(
board=[[KeyboardButton(text="Поділитись номером", request_contact=True)]],
ize_keyboard=True

essage.answer("Відправте ваш номер телефону", reply_markup=kb)
tate.set_state(RegistrationStates.waiting_for_phone_number)
```

ДОДАТОК Б. Продовження

```
ists(tid):
uests.get(f'{API_BASE_URL}users/get/{tid}/')
r.status_code == 200


user(**kwargs):
m_id = kwargs.pop('telegram_id', None)
telegram_id:
se ValueError("telegram_id is required for update")

uests.patch(f'{API_BASE_URL}users/update/{telegram_id}/', data=kwargs)
r.status_code == 200


user(telegram_id):
uests.post(f'{API_BASE_URL}users/verify/', data={'telegram_id': telegram_id})
r.status_code == 200

ля клавиатура
nu():
ReplyKeyboardMarkup(
board=[
 [KeyboardButton(text="Пралка 🧺"), KeyboardButton(text="Профіль 👤")],
 [KeyboardButton(text="Преміум 💎")],

ize_keyboard=True


on = InlineKeyboardMarkup(
keyboard=[[InlineKeyboardButton(text="Графік на прання", web_app=WebAppInfo(url="https://www.pralka.pp.ua/webapp"))]]
```

```
lineKeyboardMarkup(
ine_keyboard=[
 [
    InlineKeyboardButton(text="✅ Прийнято до уваги", callback_data=f"support_broken_ack_{message.from_user.id}")
 ]


ot.send_message(
PORT_ADMIN_CHAT_ID,
tion,
ly_markup=kb

essage.answer("Повідомлення відправлено адміністрації.")
tate.clear()

k_query(F.data.startswith("support_accept_"))
upport_accept(callback: CallbackQuery):
nt(callback.data.split('_')[-1])
ot.send_message(uid, "Ваша скарга прийнята та опрацьована. Дякуємо за звернення!")
allback.answer("Користувача повідомлено.")
allback.message.edit_reply_markup(reply_markup=None)

k_query(F.data.startswith("support_reject_"))
upport_reject(callback: CallbackQuery, state: FSMContext):
nt(callback.data.split('_')[-1])
tate.set_state(SupportStates.waiting_for_reject_reason)
tate.update_data(reject_user_id=uid, admin_msg_id=callback.message.message_id)
allback.message.answer("Вкажіть причину відмови у скарзі.")
allback.answer()

(SupportStates.waiting_for_reject_reason)
upport_reject_reason(message: Message, state: FSMContext):
await state.get_data()
ata.get("reject_user_id")
```