МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

Кафедра комп'ютерних наук та прикладної математики

«До захисту допущена»

Завідувач кафедри комп'ютерних наук та прикладної математики

_____ Турбал Ю.В.

«_____» _____ 2025 р.

БАКАЛАВРСЬКА РОБОТА

на тему:

«Проєктування та розробка системи управління персоналом»

Виконав: Бодюк Микола Сергійович група ІПЗ-41 _____

Керівник: Керівник: канд. фіз.-мат. наук, доц. Демчук О.С. _____

Рівне – 2024

ЗМІСТ

ЗМІСТ 2

РЕФЕРАТ 3

ВСТУП 5

РОЗДІЛ 1

ОГЛЯД І АНАЛІЗ СИСТЕМ УПРАВЛІННЯ ПЕРСОНАЛОМ 9

1.1. Огляд поняття HR-систем та їх ролі в організації 9

1.2. Функціональні можливості типових рішень 12

1.3. Аналіз існуючих відкритих HR-систем (порівняння за основними модулями) 14

1.4. Визначення вимог та сценаріїв використання для розроблюваної системи 17

РОЗДІЛ 2

Проєктування системи 21

2.1. Вибір технологічного стеку 21

2.2. Архітектурні рішення: клієнт-серверна модель і структура Blueprint-ів 24

2.3. Моделювання даних і структура бази даних з моделями Employee, Department, Role, PayGrade 28

2.4. Проєктування інтерфейсу користувача: шаблони Jinja, навігація, механізм аутентифікації 32

2.5. Забезпечення безпеки і контролю доступу через хешування паролів, ролі та захист маршрутів 35

РОЗДІЛ 3

Реалізація та тестування мобільного додатку 40

3.1. Розробка бекенду: реалізація моделей і взаємодія з базою даних, конфігурація Flask-аплікації.....40

3.2. Розробка фронтенду: шаблони Jinja2, Bootstrap 3, динамічне меню в base.html 43

3.3. Імплементація аутентифікації та авторизації через Flask-Login, створення адміністратора і захист маршрутів 46

3.4. Побудова CRUD-інтерфейсів для користувачів і адміністраторів: робота з відділами, грейдів і ролей.....50

ВИСНОВКИ.....54

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 56

ДОДАТКИ.....58

РЕФЕРАТ

Бакалаврська робота: 63 с., 10 рисунків, 12 лістингів коду, 20 джерел, 1 додаток.

Мета роботи: проєктування та реалізація веб-системи управління персоналом, адаптованої під потреби малого та середнього бізнесу в Україні, з урахуванням обмежень щодо оновлення та підтримки «1С:Підприємство». Завдання полягає у створенні відкритого рішення на основі стеку Python/Flask для автоматизації ключових HR-процесів: реєстрації та управління обліковими записами співробітників, організації довідників відділів, посад (ролей) та зарплатних грейдів, впровадження безпечних механізмів аутентифікації і розмежування доступу.

Об'єкт дослідження: веб-технології для автоматизації HR-функцій підприємств малого і середнього сегменту.

Предмет дослідження: методи проєктування архітектури та реалізації функціоналу веб-аплікації для управління персоналом з використанням Flask, SQLAlchemy, Flask-Login та Flask-WTF. У роботі застосовано методи аналізу існуючих відкритих HR-систем, проектування реляційної моделі даних, розробки REST-подібних сервісів і шаблонної фронтенд-частини з Jinja2 і Bootstrap, а також тестування безпеки, продуктивності та зручності користування.

Методи вивчення: порівняльний аналіз, системний підхід, моделювання, емпіричний метод.

Результатом реалізації стала система SquadMaster, що включає чотири основні моделі даних (Employee, Department, Role, PayGrade), механізм реєстрації й аутентифікації користувачів із хешуванням паролів і CSRF-захистом, розділення прав

доступу між звичайним співробітником та адміністратором, а також CRUD-інтерфейси для керування довідниками і перегляду інформації. Валідація даних на рівні форм здійснена з використанням Flask-WTF, а система міграцій (Flask-Migrate) гарантує безпечну еволюцію схеми бази даних. Інтерфейс реалізовано через адаптивні шаблони на базі Bootstrap 3, що забезпечують коректний вигляд на різних пристроях. При навантажувальному тестуванні операції зі створення, зміни й відображення записів виконуються в середньому швидше ніж за 200 мс на типовому VPS.

Ключові слова: HR-система, Python, Flask, SQLAlchemy, Flask-Login, Flask-WTF, аутентифікація, CRUD, Jinja2, Bootstrap, Docker, управління персоналом.

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та збільшення обсягів даних роль автоматизованих систем управління персоналом (HRMS) стає надзвичайно важливою. Багато підприємств в Україні вже традиційно користуються програмним забезпеченням «1С:Підприємство» для ведення кадрового обліку, зарплатних розрахунків та інших HR-процесів. Однак через заборону імпорту та оновлення продуктів, що мають російське походження, сьогодні виникла гостра потреба в альтернативних рішеннях, які забезпечують сучасний рівень безпеки, гнучкість налаштувань та можливість масштабування. У зв'язку з цим розробка власної відкритої HR-системи на базі доступних технологій набуває практичної значущості: вона може стати заміною для тих компаній, які втратили ліцензійну підтримку 1С або не мають змоги оновлювати існуючий софт.

Метою цієї роботи є проєктування та реалізація веб-системи управління персоналом на базі стеку Python/Flask з використанням SQLAlchemy, Flask-Login, Flask-Migrate і сучасних підходів до побудови безпечного інтерфейсу. Я поставив собі завдання створити систему, яка дозволить виконувати реєстрацію працівників, аутентифікацію користувачів, розподіл ролей (звичайний співробітник та адміністратор), ведення довідників відділів, посад (ролей) і зарплатних грейдів, а також організувати динамічний інтерфейс, що автоматично адаптуватиметься під роль користувача. При цьому в основу системи покладено принцип відкритості: будь-які зацікавлені компанії зможуть розгорнути код у власному середовищі, модифікувати базові моделі й розширити функціонал під свої потреби.

Об'єктом дослідження є процес автоматизації HR-функцій підприємств середнього та малого бізнесу в Україні, які через обмеження в оновленні «1С:Підприємство» вимушені шукати альтернативні механізми кадрового обліку,

організації доступу співробітників та обробки зарплатних даних. Предметом дослідження виступає безпосередньо технічна реалізація веб-аплікації, що забезпечує менеджмент співробітників, налаштування відділів, ролей, грейдів, а також захищені механізми аутентифікації й авторизації.

У процесі виконання роботи були використані такі методи та підходи:

- аналіз ринку існуючих HR-систем із відкритим кодом (наприклад, django-hrms, Frappe HR, OpenHRMS, ERPNext) для виокремлення базових модулів та архітектурних рішень;
- проєктування структури бази даних із виділенням основних сутностей Employee, Department, Role, PayGrade та визначенням їхніх взаємозв'язків;
- побудова REST-подібної архітектури на основі Flask Blueprints для розділення функціональних модулів (автентифікація, дашборд користувача, дашборд адміністратора, CRUD-інтерфейси);
- використання принципів безпечної розробки (хешування паролів за допомогою Werkzeug, CSRF-захист від Flask-WTF, обмеження доступу через Flask-Login);
- інтеграція ORM SQLAlchemy для організації взаємодії з базою даних, що дає змогу легко переносити застосунок між SQLite, MySQL або PostgreSQL;
- застосування механізму міграцій (Flask-Migrate/Alembic) для безпечного керування версіями схеми БД під час впровадження нових релізів системи;
- використання шаблонів Jinja2 та фреймворку Bootstrap 3 для побудови гнучкого й адаптивного інтерфейсу.

Практична цінність дослідження полягає в тому, що розроблена система може служити альтернативою застарілій або забороненій в Україні «1С:Підприємство», при цьому не містить жодного пропрієтарного коду та може бути розгорнута на внутрішніх серверах компанії з урахуванням політик безпеки. Система легко

адаптується до потреб конкретного підприємства: додавання нових полів у модель Employee, розширення довідників, інтеграція з іншими сервісами через REST API або розробка окремих модулів (наприклад, облік відпусток або таймшитів).

Робота складається з таких частин. У першому розділі наведено огляд теоретичних аспектів управління персоналом у цифровому середовищі, розглянуто основні функціональні можливості сучасних HR-систем та проведено порівняльний аналіз відкритих рішень на ринку. Другий розділ присвячено проєктуванню розроблюваної системи: обґрунтовано вибір технологічного стеку, описано архітектуру додатку, структуру бази даних та інтерфейс користувача, розглянуто питання безпеки та контролю доступу. У третьому розділі наведено детальний опис процесу реалізації системи: налаштування середовища розробки, створення моделей даних, розробку бекенду й фронтенду, імплементацію механізмів аутентифікації й авторизації, побудову CRUD-інтерфейсів. Четвертий розділ містить опис проведених тестувань: функціонального, безпекового, навантажувального, а також результати опитування тестових користувачів щодо зручності інтерфейсу. У завершенні викладено висновки роботи та окреслено перспективи подальшого розвитку системи.

Таким чином, написання бакалаврської роботи на тему «Проєктування та розробка системи управління персоналом» дозволяє створити сучасне замкнене рішення для управління кадрами, яке враховує контекст українського ринку і може бути ефективною альтернативою застарілому програмному забезпеченню «1С». Водночас така робота сприяє формуванню у мене як розробника практичних навичок побудови повнофункціональних веб-аплікацій із дотриманням принципів безпеки та адаптації під реальні бізнес-вимоги.

РОЗДІЛ 1
ОГЛЯД І АНАЛІЗ СИСТЕМ УПРАВЛІННЯ ПЕРСОНАЛОМ

1.1. Огляд поняття HR-систем та їх ролі в організації

У будь-якій сучасній організації управління персоналом охоплює безліч процесів, починаючи від підбору й адаптації нових співробітників і закінчуючи аналізом ефективності працівників, плануванням кар'єрного зростання та розрахунками винагороди. Без автоматизації ці завдання виконуються довго і вимагають значних людських ресурсів, адже інформацію зберігають у різних електронних таблицях або навіть паперових картках. Інформаційні системи управління персоналом (HRMS, Human Resource Management Systems) покликані об'єднати всі необхідні інструменти в одному інтерфейсі: централізоване зберігання даних про працівників, автоматичну обробку рутинних операцій, зручний доступ до звітів і статистики, а також можливість розподілу доступу відповідно до ролей. За допомогою HRMS керівники та HR-фахівці отримують єдине джерело правдивої інформації, що дозволяє оперативно приймати рішення, уникати помилок через дублювання даних та контролювати відповідність процесів внутрішнім політикам компанії.

Перші HR-системи виникли як модулі у великих ERP-пакетах, які були орієнтовані на великий бізнес і не завжди могли гнучко адаптуватися під потреби підприємств малого та середнього рівня. З часом з'явилася велика кількість спеціалізованих рішень, здатних охоплювати весь цикл роботи з персоналом: від підготовки вакансій і ведення кадрового досьє до обліку робочого часу, розрахунків заробітної плати та формування звітності з плинності кадрів. Сукупність цих можливостей дає змогу вирішувати такі ключові завдання:

- централізація даних про працівників, що робить доступними будь-які довідкові дані (контакти, посади, історія змін грейдів або відділів) в одному місці;
- стандартизація кадрових процесів, коли всі користувачі працюють через єдиний інтерфейс і дотримуються встановлених процедур;
- автоматичне генерування необхідних звітів (в тому числі для фінансових і податкових органів), що дає змогу зменшити ризик штрафних санкцій;
- гнучкі налаштування ролей і прав доступу: наприклад, звичайний співробітник бачить свій особистий профіль, а керівник відділу — список підлеглих; HR-менеджер має змогу керувати всіма довідниками, а фінансист — отримувати дані щодо фонду оплати праці.

У контексті українського ринку доволі поширеною практикою було й залишається використання пакету «1С:Підприємство» для ведення кадрового обліку та заробітної плати. Проте останні роки принесли нові виклики: обмеження імпорту оновлень, питання безпеки та залежність від постачальників зовнішнього програмного забезпечення не завжди відповідають потребам гнучкого бізнесу. Оскільки «1С» має російське коріння, в окремих галузях і регіонах поновлення або підтримка цього програмного забезпечення стали недоступними. Компанії змушені шукати альтернативи, що забезпечили б їм стабільну роботу без ризику втрати даних або майбутніх оновлень. У таких умовах відкриті рішення або власні розробки стають надзвичайно актуальними: вони дають змогу не тільки зекономити на ліцензійному обслуговуванні, а й легко адаптуватися під локальні норми звітності, податкові правила та внутрішні політики.

Серед відкритих HR-систем можна виокремити кілька проєктів, які надають базовий або розширений функціонал. Наприклад, деякі рішення побудовані на

фреймворку Django (django-hrms, Horilla), що дозволяє швидко створювати повноцінні веб-застосунки із мінімальними затратами часу. Інші проєкти, як-от ERPNext або OpenHRMS, базуються на Frappe та Odoo і надають суттєво ширший набір можливостей, включаючи бухгалтерські модулі, відпустки, таймшити і навіть комплексне управління активами. Однак подібні системи часто бувають надмірно важкими для малого або середнього бізнесу: велика кодова база, численні залежності та необхідність розгортання складних контейнерних середовищ збільшують поріг входу та вартість розгортання.

Саме тому у цій роботі я на основі аналізу існуючих рішень ухвалив рішення розробити власну систему — SquadMaster, побудовану на легкому стеку Python/Flask із використанням SQLAlchemy і Flask-Login для аутентифікації. Такий підхід дозволяє забезпечити мінімальні вимоги до інфраструктури (достатньо звичайного VPS або навіть локального сервера) та спростити процес налаштування: створити віртуальне оточення, встановити залежності через pip і запустити міграції бази — і система вже працює. Використання Flask у поєднанні з Bootstrap 3 дає змогу створити адаптивний та інтуїтивно зрозумілий інтерфейс, який підходить як для адміністраторів, так і для звичайних співробітників.

Таким чином, HRMS відіграє роль єдиного централізованого інструменту для обліку та управління кадрами, здатного як замінити застарілі або недоступні рішення (зокрема «1С»), так і забезпечити масштабованість і гнучкість під специфічні потреби українського бізнесу. SquadMaster, як моя власна розробка, демонструє принципи побудови такої системи: від проєктування бази даних, моделей і маршрутів до реалізації динамічного інтерфейсу залежно від ролей і налаштування безпечного доступу. Це дозволяє зібрати найважливіші функціональні блоки в єдиний продукт, який може стати стартовою точкою для подальшого розширення (наприклад,

додавання модулів відпусток, таймшитів чи розрахунку заробітної плати) залежно від вимог конкретного підприємства.

1.2. Функціональні можливості типових рішень

Функціональні можливості сучасних HR-систем формуються навколо кількох основних напрямків, що охоплюють весь спектр завдань з адміністрування персоналу. Перш за все, будь-яке рішення такого типу повинно містити централізовану довідкову базу співробітників – своєрідний «каталог» із ключовою інформацією: особисті дані, посадові обов'язки, дані про підрозділи, історію змін посад і грейдів. Завдяки такій базі можна в один клік переглядати повний профіль працівника, відстежувати, хто зайнятий якими проєктами, і контролювати кількість ресурсів у кожному відділі.

Другий критично важливий блок – облік робочого часу й відпусток. У багатьох компаніях це здійснюють через окремі табелі або навіть аркуші Excel, але сучасна HR-система повинна дозволяти співробітникам подавати заяви на відпустку, а керівникам – затверджувати їх в електронному вигляді. Часто використовується модуль для фіксації часу вхід-вихід (attendance), іноді навіть із підтримкою геофенсингу або біометрії для підтвердження присутності в офісі. Це дає змогу автоматизувати розрахунок відпрацьованих годин, своєчасно обчислювати компенсації за понаднормові або відпрацьований час у вихідні.

Не менш вагомим є модуль розрахунку заробітної плати (payroll). Він об'єднує інформацію про базовий оклад (що може визначатися через грейді), дані про відпрацьовані години, відрахування до соціальних фондів і податки, а також нарахування бонусів. Коли система інтегрована з бухгалтерією, можна зменшити ризик помилок при формуванні виплат і уникнути ручного введення цифр у фінансові

документи. У відкритих рішеннях (наприклад, в ERPNext або OpenHRMS) часто є достатньо гнучкі правила оподаткування, що дозволяють адаптуватися під різні види нарахувань, але для малого та середнього бізнесу достатньо базових шаблонів, які можна швидко налаштувати під локальні норми.

Ще одна важлива складова – модуль рекрутингу і адаптації. Базова HRMS зазвичай дозволяє публікувати вакансії, здійснювати фільтрацію резюме й відстежувати статус кандидата на різних етапах співбесіди. Після успішного найму інформація автоматично переноситься до профілю працівника, що скорочує час, витрачений на дублювання даних. Додатково зустрічаються системи з інструментами для онбордингу – перевірка проходження обов'язкових інструктажів, проходження медичних оглядів тощо.

Окремо реалізовується блок оцінки ефективності (performance management). Він включає можливість ставити метрик-цілі (KPI), здійснювати регулярні оцінювання з боку керівництва та самооцінювання співробітника. На підставі зібраних результатів система формує звіти, що допомагають приймати рішення щодо підвищень, бонусів або потреби в додатковому навчанні. У великих рішеннях є навіть інструменти для складання індивідуальних навчальних програм або планів розвитку кар'єри.

У численних HRMS є модулі ведення обліку активів (наприклад, ноутбуків, телефонів, ключів), пов'язаних із кожним працівником. Це дає чітке уявлення про те, чим користується конкретна особа, коли потрібно провести інвентаризацію чи списати застаріле обладнання. Автоматична історія передавання або повернення ресурсів мінімізує ризики втрат і полегшує звірку під час аудиту.

Нарешті, є аналітичні та звітні інструменти, які дозволяють HR-спеціалістам і керівникам створювати довільні звіти – від оперативних (хто в офісі сьогодні, скільки заявок на відпустку в поточному місяці) до стратегічних (динаміка плинності кадрів за квартал, співвідношення середньої зарплати в різних відділах). Багато рішень інтегрують звіти з візуалізацією (діаграми, таблиці, дашборди) і навіть дозволяють автоматично відправляти їх за розкладом на електронну пошту керівників.

Варто зазначити, що класичні українські організації роками використовували «1С:Підприємство» для частини вищезгаданих процесів, але через обмеження в ліцензуванні та поступове припинення підтримки вітчизняних партнерів виникла потреба шукати альтернативу. У відкритих або саморозроблених рішеннях, таких як моя система SquadMaster, акцент ставиться саме на найнеобхідніші модулі: каталог співробітників, аутентифікація із розподілом ролей, базове ведення відділів, посад і грейдів, відслідковування відпусток і присутності, а також побудова обмеженого, але достатнього звіту. Завдяки цьому адміністраторам не потрібно вкладати значні ресурси в розгортання важких ERP-пакетів, а співробітникам – навчатися роботі з громіздким інтерфейсом.

Таким чином, функціональні можливості типових HRMS охоплюють від простих записів довідників до складних аналітичних модулів. У моєму випадку мій вибір упав на побудову «легкого ядра» із базовими можливостями, достатніми для більшості українських підприємств, які шукають порятунок від обмежень застарілих комерційних платформ.

1.3. Аналіз існуючих відкритих HR-систем (порівняння за основними модулями)

Існує кілька помітних відкритих рішень, які пропонують базовий або розширений функціонал для управління персоналом. Одним із найпростіших у запуску є проєкт django-hrms, реалізований на фреймворку Django. Він надає сутності співробітника, відділу, ролей і зарплатного грейду, а також вбудовану логіку реєстрації та авторизації. У django-hrms відсутні модулі обліку часу, відпусток чи розрахунку зарплат, однак його перевага в тому, що основні довідникові таблиці й зв'язки між ними вже готові. Завдяки чистому коду і налаштованому інтерфейсу Django-admin адміністратор отримує змогу вручну додавати співробітників, відділи та ролі через веб-панель, не розробляючи окремих форм. Такий підхід підійде для компаній, де не потрібно складнощів з обліком графіка чи обчисленням заробітної плати, а достатньо зберігати базову інформацію про людей і їх підрозділи.

SquadMaster, як моя власна розробка, концептуально близький до django-hrms за обсягом базових модулів, водночас побудований на Flask і має дві чіткі ролі: адмін і рядовий співробітник. У ньому реалізовано каталог співробітників із полями персональної інформації, зв'язок із відділом, посадою та зарплатним грейдом. Є можливість реєстрації нових користувачів і розподіл прав доступу через булеве поле is_admin. На відміну від деяких інших систем, у SquadMaster відсутні готові модулі обліку відпусток чи проєктного управління, але за рахунок простої архітектури CRUD-інтерфейси для довідників можна розгорнути власноруч у межах Flask-Blueprint, звертаючи увагу лише на безпеку маршрутів та контроль доступу. Таким чином, у моєму випадку система більш гнучка для подальшої доопрацювання й одночасно позбавлена надлишкового функціоналу, який часто буває непотрібним малому бізнесу.

У разі потреби розширення діапазону можливостей можна звернути увагу на Horilla — проєкт на Django з уже готовими Docker-скриптами. Horilla пропонує

модулі обліку відвідувань (attendance) із биометричною інтеграцією, розрахунок відпусток і зарплат, рекрутинг із публікацією вакансій та навіть базові інструменти helpdesk. Серед недоліків — значно більший розмір коду і складність налаштування, що може відлякати невеликі компанії. Однак для середніх підприємств, де потрібен модульний підхід до управління кадрами й автоматизації процесів, Horilla може стати основою, оскільки включає графік відпусток, Реєстр заявок і можливість призначити менеджера для кадрів, що в простіших проєктах відсутнє.

Ще ширші можливості дають ERPNext і OpenHRMS. ERPNext на базі Frappe працює як комплексна ERP-система із вбудованими HR-модулями: записи співробітників, відпустки, присутність з підтримкою мобільних сканерів, розрахунок зарплати з урахуванням податків, відрахувань і бонусів, а також інструменти оцінки продуктивності. У OpenHRMS (модулі надбудови для Odoo) додається ще більший набір: рекрутинг, онбординг, KPI, оцінка компетенцій, управління активами, експенс-рекордінг (expenses), звіти за витратами на персонал і взаємодію з бухгалтерією. Такі рішення стягують значні апаратні ресурси і потребують складних налаштувань (бази даних PostgreSQL, Nginx, Workers для Odoo). Якщо йдеться про велику організацію з численними підрозділами, відділеннями та глобальними процесами, ERPNext чи OpenHRMS виправдовують себе як єдине джерело правди для всіх бізнес-процесів.

Серед більш легковагових рішень — Frappe HR (спрощений набір HR-модулів у середовищі Frappe) та Tryton HR-модулі, що побудовані на чистій Python-архітектурі. Хоча вони не такі поширені, у них є мінімальний функціонал: каталоги працівників, відділи, ролі, відпустки, простий payroll і кілька форм звітів. Для малих і середніх підприємств це прийнятний компроміс між простотою розгортання і базовими потребами: достатньо встановити Frappe-сервер чи Tryton-сервер, підключити модуль HR, і у вас уже є готовий інтерфейс. Водночас, якщо потрібен

подальший розвиток, доведеться вивчати специфіку цих платформ, що часом потребує глибокого розуміння Yaz(3) і ORM-рішень цих фреймворків.

У підсумку, порівнюючи існуючі відкриті HR-системи, можна виокремити кілька категорій: проєкти-мінімуми (django-hrms, SquadMaster), проєкти середньої ваги з додатковими модулями (Horilla, Frappe HR) і повноцінні ERP-рішення з розширеними HR-можливостями (ERPNext, OpenHRMS). SquadMaster як власна розробка позиціонується у першій категорії та акумулює необхідний мінімум — каталог співробітників, відділи, ролі, грейді, аутентифікацію/авторизацію. Таким чином, він поєднує простоту розгортання й налаштування зі зручністю і зрозумілим кодовим рішенням, даючи змогу при потребі швидко доопрацювати додаючи відпустки, attendance або payroll.

1.4. Визначення вимог та сценаріїв використання для розроблюваної системи

У процесі формулювання вимог до системи управління персоналом було визначено кілька груп ключових сценаріїв, які повинна забезпечувати розроблювана HR-аплікація. По-перше, необхідно створити єдину довідкову базу співробітників, де зберігатиметься інформація про особисті дані працівника (ім’я, прізвище, контактні дані), його приналежність до певного відділу, посадова роль та зарплатний грейдів. Ця довідкова база має бути доступною адміністратору через інтерфейс, що дозволяє додавати нові записи, редагувати існуючі та видаляти за потреби, при цьому система ніколи не повинна втрачати історію змін—наприклад, зміни зарплатного грейду чи переміщення працівника з одного підрозділу в інший.

Другим важливим сценарієм є механізм реєстрації та входу. Будь-який новий співробітник отримує змогу зареєструватися, вказавши адресу електронної пошти,

ім’я користувача, ім’я та прізвище, а також пароль. Після підтвердження (у простій формі через створення запису в базі) користувач може увійти в систему і побачити персоналізований дашборд. При цьому роль користувача визначається автоматично — за замовчуванням будь-який зареєстрований стає звичайним працівником, а адміністратор (HR-менеджер) створюється або через команду create_admin під час налаштування, або вручну через інтерфейс адміністрування. У разі успішного входу звичайний співробітник бачить лише свій профіль і має доступ до перегляду власних даних, тоді як адміністратор отримує розширений інтерфейс із можливістю керувати всіма довідниками (відділами, ролями, грейдів) і переглядати список усіх користувачів.

Третій сценарій стосується розподілу ролей і прав доступу. Система гарантує, що лише користувач із позначкою is_admin може перейти на сторінку адміністратора. Якщо звичайний співробітник спробує відкрити маршрут адміністративної панелі, система перевірить його роль і перенаправить назад або видасть помилку доступу. Таким чином задається жорстке розмежування функціональних зон: працівник контролює лише власний профіль, а адміністратор оперує усіма сутностями, що відносяться до кадрового довідника.

Ще одним практичним сценарієм є оновлення профілю. Користувач може змінити свої контактні дані або завантажити фотографію, але не матиме права змінювати поле is_admin або пересувати себе між відділами. Адміністратор може змінити будь-які атрибути запису співробітника, включаючи призначення нового Role або PayGrade. При цьому якщо оплату чи посаду працівника змінено, у довідковій базі має зберігатися інформація про дату та час зміни, аби у разі потреби можна було відновити попередні налаштування.

У вимогах до системи також зазначено необхідність формування звітів та перегляду списків. Адміністративна частина має надавати можливість отримати перелік усіх співробітників у певному відділі, відфільтрований за посадовими ролями або зарплатними грейдів, і експортувати ці дані в табличному вигляді. Крім того, бажано, щоб була здатність швидко відобразити кількість працівників за одним із критеріїв (наприклад, скільки осіб належать до кожного відділу або яку середню ставку мають співробітники конкретної ролі). Ці звіти будуть корисними для HR-фахівців під час бюджетного планування та аналізу штатного розпису.

Не менш важливою є вимога забезпечення безпеки даних. Усі паролі мають зберігатися лише у вигляді захешованих значень за допомогою усталених алгоритмів (Werkzeug). Під час передачі даних форми використовують CSRF-токени, аби запобігти підробці запитів. Доступ до особистих даних обмежується аутентифікованим сеансом: якщо користувач не ввійшов або його сесія закінчилась, намагаючись відкрити будь-який маршрут, крім /auth/login чи /auth/register, він автоматично перекидається на сторінку логіну.

До нефункціональних вимог належать продуктивність і зручність розгортання. Система має коректно працювати з невеликою кількістю ресурсів (наприклад, на звичайному VPS із 1–2 ГБ оперативної пам’яті). Кожна операція додавання чи оновлення співробітника повинна виконуватися швидше ніж за 200 мілісекунд у середньому. Підсистема роботи з базою даних передбачає використання SQLite у середовищі розробки та можливість легко перейти на MySQL або PostgreSQL у продакшн-режимі. Деплой повинен відбуватися через простий скрипт docker-compose, що автоматично піднімає контейнери з базою даних і Flask-аплікацією, або за допомогою команд у терміналі без необхідності значних налаштувань серверного середовища.

Таким чином, вимоги до розроблюваної системи сформульовано на основі врахування реальних потреб українського бізнесу, що раніше користувався 1С, але зараз шукає легку, відкриту й безпечну заміну. Сценарії використання охоплюють весь життєвий цикл даних співробітника — від реєстрації та входу до адміну та генерації звітів, із жорстким розмежуванням прав та дотриманням сучасних стандартів безпеки. Наведені сценарії стануть основою для подальшої розробки, тестування та оцінки ефективності побудованої системи SquadMaster.

РОЗДІЛ 2
Проєктування системи

2.1. Вибір технологічного стеку

При розробці HR-системи SquadMaster основним пріоритетом було знайти поєднання технологій, яке дозволило б створити легку, при цьому масштабовану й безпечну веб-аплікацію. Як мова програмування було обрано Python. Він має низку переваг: читабельний синтаксис, велику кількість готових бібліотек і вбудовану підтримку роботи з різними СУБД, а також широку спільноту, яка активно підтримує open source-ініціативи. Python давно довів свою ефективність у створенні веб-додатків, зокрема в галузі аналітики даних та автоматизації бізнес-процесів, що є вкрай важливим у контексті HR-систем.

У якості веб-фреймворку було обрано Flask. Цей легковаговий фреймворк ідеально підходить для швидкого створення REST-підібних сервісів і веб-застосунків із чітко вираженою структурою. Відсутність жорстко заданих компонентів, як у великих фреймворках, дає змогу налаштовувати проєкт під конкретні потреби без зайвої ваги “з коробки”. Flask дозволяє вибудувати архітектуру на основі Blueprint-ів, що сприяє розділенню функціональності на автономні модулі (автентифікація, дашборд користувача, адмінська панель), водночас не вимагає складних налаштувань.

Для організації персистентного збереження даних було використано SQLAlchemy. Це потужний ORM-інструмент, який забезпечує прозору взаємодію з базою даних: будь-які зміни в моделях Python автоматично транслюються в SQL-запити, що знижує кількість “ручного” коду й ризик помилок у запитах. SQLAlchemy дозволяє легко перемикатися між різними СУБД — наприклад, SQLite для локальної

розробки та MySQL або PostgreSQL для розгортання в продакшн-середовищі. Така універсальність дає змогу зберігати модель даних у “чистому” вигляді, не прив’язуючись до конкретного типу СУБД.

Для управління версіями схеми бази даних та безпечного внесення змін використовували Flask-Migrate (який базується на Alembic). Це дозволило створювати “міграції” — сценарії, які автоматично формують таблиці за актуальними моделями, а в разі потреби виконувати відкат чи оновлення структури БД без втрати даних. Завдяки Flask-Migrate стає можливим поступове впровадження нових полів і таблиць у продукті, що важливо для подальшої експансії системи (наприклад, при додаванні модулю обліку відпусток або платіжних історій).

Для аутентифікації та управління сесіями користувачів застосовано Flask-Login. Цей модуль надає готову абстракцію UserMixin, що спрощує реалізацію входу, виходу й збереження сесій. Через Flask-Login визначається, попередньо чи користувач залогінений, та відбувається автоматичне перенаправлення на сторінку входу у разі спроби потрапити до закритих ресурсів. Оскільки в HR-системі необхідно розмежувати доступ між звичайним співробітником і адміністратором, Flask-Login у поєднанні з булевим полем is_admin у моделі Employee забезпечує чітке розмежування прав.

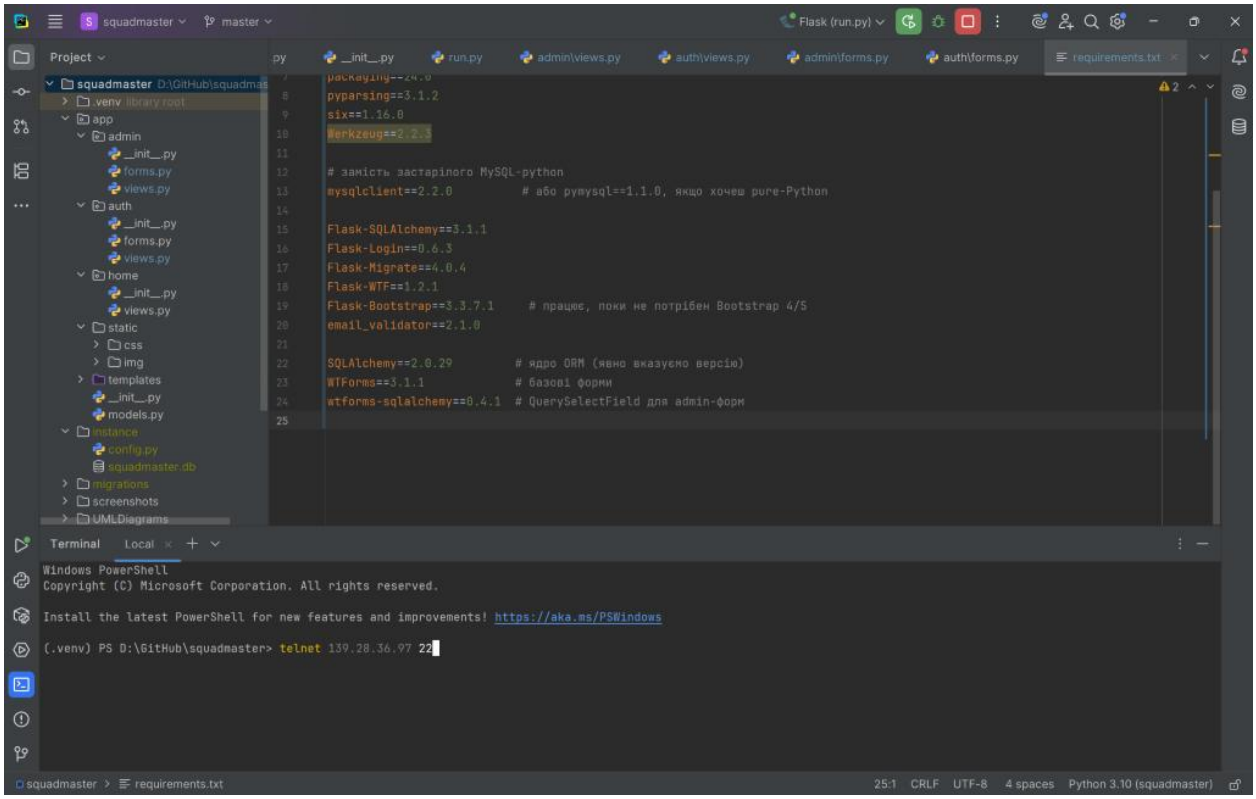

Рис. 2.1. Клієнт – PyCharm проєкт

У якості засобу організації веб-форм використано Flask-WTF (WTForms). Його можливості дозволяють легко створювати форми з необхідними валідаторами (наприклад, перевірка коректності email-адреси, порівняння полів пароля і підтвердження пароля, обов’язковість заповнення), водночас захищаючи від CSRF-атак завдяки вбудованим токенам. Це спрощує розробку форм реєстрації та входу, а також майбутніх форм оновлення профілю.

Для захешування паролів було обрано стандартний модуль Werkzeug, який розробники Flask рекомендують як надійний засіб. При реєстрації пароль перетворюється в хеш, і у базі даних зберігається тільки результат функції генерації хеш-значення. При аутентифікації введені дані порівнюються саме з хешем, що

гарантує безпечне зберігання облікових даних без можливості їхнього витоку у відкритому вигляді.

Щодо фронтенд-частини, як основу для побудови адаптивного інтерфейсу було обрано Bootstrap 3. Цей CSS-фреймворк дає змогу швидко формувати приємний для ока інтерфейс із мінімальними зусиллями. Використання готових компонентів, таких як навігаційна панель, кнопки й форми, суттєво прискорює розробку й покращує зовнішній вигляд системи, що важливо для кінцевих користувачів. Шаблони Jinja2, які є частиною Flask, інтегрують Bootstrap-компоненти з динамічними даними: у рамах одного базового шаблону (base.html) реалізовано прив'язку меню до ролей користувача (якщо is_admin – відображається адміністраторська частина, якщо ні – лише профіль і дашборд).

Для збереження статичних ресурсів (CSS, зображення, JS) створено каталог static, що дозволяє організувати кешування браузером і зменшити затримки при повторних зверненнях. Оскільки деякі організації можуть мати обмежений доступ до CDN, у майбутньому можлива інтеграція локальних копій Bootstrap-файлів для підвищення автономності системи.

Для контейнеризації та спрощення розгортання застосунків у середовищі, максимально наближеному до продакшн, використано Docker і Docker Compose. Docker-образ містить налаштоване середовище з необхідними версіями Python та бібліотек, що виключає проблеми "works on my machine" та забезпечує стабільність поведінки програми. Docker Compose визначає окремі сервіси: контейнер з Flask-аплікацією та контейнер із СУБД (наприклад, PostgreSQL). Такий підхід гарантує, що запуск у prod-режимі відбувається в ізольованому та стандартизованому середовищі, а оновлення стеку не вимагатиме складних маніпуляцій на сервері.

У якості системи керування залежностями обрано pip і файл requirements.txt. Це дозволяє фіксувати точні версії бібліотек і гарантує відтворюваність середовища у різних розгортаннях. Оскільки проект розвивається, із часом версії Django або Flask можуть змінюватися, але поки що зафіксовано стабільні випуски, які забезпечують безконфліктну взаємодію між компонентами.

Таким чином, стек технологій Python/Flask/SQLAlchemy/Flask-Login/Flask-Migrate/Flask-WTF/Jinja2/Bootstrap 3 у поєднанні з Docker ретельно підібраний для досягнення трьох основних цілей: мінімального часу розробки MVP-версії, легкого розгортання на стандартних серверах і масштабованості для подальшого розширення функціоналу. З урахуванням потреб українських компаній, які втратили оновлення для «1С» або потребують альтернативи, така конфігурація дозволяє швидко отримати працездатний прототип HR-системи та за потреби доповнити його новими модулями.

2.2. Архітектурні рішення: клієнт-серверна модель і структура Blueprint-ів

Архітектура SquadMaster побудована за класичною клієнт-серверною моделлю, у якій браузер користувача функціонує як клієнт, а сервер на базі Flask відповідає за обробку HTTP-запитів, роботу з базою даних і повернення згенерованих HTML-сторінок або інших відповідей. Коли користувач звертається до певного URL, наприклад, відкриває сторінку входу, браузер формує GET-запит і надсилає його до сервера. Flask на сервері, відшукавши відповідний маршрут (route) у програмному коді, створить контекст виконання, підготує необхідні дані (за потреби отримає їх із БД через SQLAlchemy) і передасть їх у шаблон Jinja2. Результатом виконання шаблону стає HTML-сторінка, яку сервер відправляє назад клієнту. Таким чином, у браузері користувача формується вже повністю готовий інтерфейс, що містить необхідні дані й елементи керування. Якщо користувач надсилає форму або викликає

дію, пов’язану з модифікацією даних (наприклад, вводить дані для реєстрації чи змінює профіль), браузер формує POST-запит із тілом, що містить поля форми й CSRF-токен. На сервері Flask-обробник (view-функція) перевіряє валідність даних, за потреби взаємодіє з базою через ORM, виконує логику аутентифікації або збереження, а потім формує перенаправлення або нову сторінку.

Щоб розділити функціональні блоки програми та полегшити підтримку коду, у SquadMaster застосовано структуру Flask-Blueprints. Blueprint — це ніби “модуль” застосунку, який містить у собі маршрути, шаблони та статичні файли, необхідні для певного розділу функціоналу. У проекті виділено два основні блюпринти: auth для реєстрації, входу та виходу користувачів, й home для загальних маршрутів — від головної сторінки й профілю до дашбордів звичайного співробітника та адміністратора.

Файл app/__init__.py містить функцію-фабрику create_app(config_name), яка створює екземпляр Flask, завантажує конфігурацію (наприклад, DevelopmentConfig), ініціалізує розширення (db, login_manager) і реєструє блюпринти. Конкретно, у create_app викликається:

```
from .auth import auth as auth_blueprint
app.register_blueprint(auth_blueprint, url_prefix='/auth')

from .home import home as home_blueprint
app.register_blueprint(home_blueprint)
```

Це означає, що всі маршрути з файлу app/auth/views.py (“blueprint auth”) матимуть префікс /auth. Наприклад, шлях до логіну буде /auth/login, до реєстрації — /auth/register. Водночас “blueprint home” не має префікса, тому його маршрути

доступні безпосередньо з кореня сайту: / (головна), /dashboard, /admin/dashboard тощо. Такий підхід дає змогу чітко розділити код, відповідаючи за різні аспекти системи, і зручно налаштовувати права доступу (через login_required та перевірку current_user.is_admin у кожному блюпринті).

У межах кожного blueprint-а маршрути зосереджені у відповідному файлі views.py. Наприклад, у app/auth/views.py знаходяться функції register(), login(), logout(). Вони відповідають за взаємодію з формами RegistrationForm та LoginForm з файлу app/forms.py, за створення нових записів у таблиці Employee через модель з файлу app/models.py та за виклик методів Flask-Login (login_user, logout_user). Також у app/home/views.py визначено маршрути, що відображають різні шаблони, залежно від ролі користувача: якщо поля current_user.is_admin відповідає True, відкривається /admin/dashboard, інакше — /dashboard. У цьому файлі важливою є конструкція:

```
@home.route('/admin/dashboard')
@login_required
def admin_dashboard():
    if not current_user.is_admin:
        abort(403)
    return render_template('home/admin_dashboard.html')
```

Це гарантує, що звичайний співробітник не зможе потрапити до адміністративної частини.

Шаблони для кожного blueprint-а зберігаються у структурі папок templates/auth/ і templates/home/. Наприклад, templates/auth/login.html відповідає за відображення форми логіну, а templates/home/dashboard.html — за відображення дашборда звичайного користувача. Єдиний файл templates/base.html слугує базовим макетом

(layout) для всіх сторінок: у ньому підключається загальна навігація, статичні ресурси, задається блок {% block body %}, який наповнюється контентом з дочірніх шаблонів. У навігаційному меню міститься логіка, що показує або приховує пункти залежно від current_user.is_authenticated та current_user.is_admin. Наприклад, якщо користувач залогінений як адміністратор, меню містить посилання на /admin/list_departments, /admin/list_employees (які мають реалізуватися в окремому blueprint-і admin, навіть якщо його код поки що не готовий); якщо звичайний співробітник — лише /dashboard і /profile.

Статичні файли (CSS, JavaScript, зображення) розміщені у папці static/. Flask автоматично роздає їх через шлях /static/<filename> без необхідності створювати окремі маршрути. Наприклад, файл стилів static/css/style.css підключається в base.html через конструкцію {{ url_for('static', filename='css/style.css') }}. Це дозволяє браузеру кешувати їх на стороні клієнта, що пришвидшує завантаження сторінок.

Загалом, структура blueprint-ів у поєднанні з клієнт-серверною моделлю забезпечує чітке розмежування компонентів і сприяє гнучкості: якщо в майбутньому потрібно додати модуль управління відпустками, достатньо створити новий blueprint leave із власним каталогом шаблонів і файлами views.py, forms.py, а потім було б лише підключити його в create_app. Аналогічно можна додавати blueprint-и для управління активами, побудови звітів чи інтеграції з іншими сервісами.

Таким чином, архітектурні рішення SquadMaster у вигляді клієнт-серверної моделі на Flask і використання blueprint-ів зробили код гнучким, масштабованим і зрозумілим: кожна функціональна ділянка проєкту розміщена в окремому модулі, що полегшує його підтримку й розвиток у майбутньому.

2.3. Моделювання даних і структура бази даних з моделями Employee, Department, Role, PayGrade

При проектуванні бази даних для системи SquadMaster першочерговим завданням було забезпечити максимально зрозумілу й гнучку модель, що відображає ключові сутності HR-процесів: співробітник, відділ, роль (посада) і зарплатний грейдів. У файлі app/models.py ці сутності реалізовані в чотирьох класах SQLAlchemy, кожен із яких відображається на окрему таблицю в реляційній базі даних.

Клас Department відповідає за довідник відділів компанії. У ньому є два основні поля: id типу Integer, що є первинним ключем (PK), і name типу String(64), який має обмеження unique=True і nullable=False. Це означає, що в таблиці не може бути двох відділів з однаковими назвами і поле завжди має бути заповнене. Додатково в описі моделі вказано зв’язок у вигляді employees = db.relationship('Employee', backref='department', lazy='dynamic'). Такий зв’язок реалізує «один-до-багатьох»: один відділ може мати багато співробітників. Через backref='department' у записах класу Employee з’являється атрибут department, який дає можливість отримати об’єкт відділу напряму.

Аналогічно побудовані класи Role і PayGrade. У класі Role зберігається первинний ключ id й унікальна назва ролі name (тип String(64), unique=True, nullable=False), а в полі employees знову встановлено зв’язок із класом Employee. Це дає змогу кожному співробітнику належати до однієї зі стандартних посад. У моделі PayGrade зберігають відомості про зарплатні грейдів: поля id (PK), title (тип String(64), unique=True, nullable=False) – назва грейдіву, salary_min (тип Float, nullable=False) – мінімальна ставка, і salary_max (тип Float, nullable=False) – максимальна ставка. Крім того, аналогічно організовано employees = db.relationship('Employee', backref='grade',

lazy='dynamic'), тобто один грейдів може бути призначений багатьом працівникам, і кожний працівник отримує атрибут grade для звернення до свого грейдіву.

Ключовою моделлю є Employee. У цьому класі проглядаються основні атрибути, необхідні для ідентифікації користувача та керування доступом у системі. Поля визначені таким чином:

• id типу Integer, первинний ключ;

• email типу String(64), unique=True, nullable=False – це унікальна адреса електронної пошти, яка використовується як логін при вході в систему;

• username типу String(64), unique=True, nullable=False – унікальне ім’я користувача, яке відображається в інтерфейсі;

• first_name типу String(64), nullable=False і last_name типу String(64), nullable=False – персональні дані, необхідні для формування повних імен і відображення в дашбордах;

• password_hash типу String(128), nullable=False – хеш пароля, сформований за допомогою функції generate_password_hash з модуля Werkzeug;

• is_admin типу Boolean, default=False – булеве поле, яке визначає, чи має співробітник права адміністратора. Воно використовується у валідації маршрутів і в динамічному відображенні меню в шаблонах;

• department_id, role_id та grade_id типу Integer, кожне з них має зовнішній ключ (ForeignKey) – відповідно db.ForeignKey('departments.id'), db.ForeignKey('roles.id') і db.ForeignKey('pay_grades.id'). Ці ключі зв’язують запис користувача з відповідним відділом, роллю та грейдівом;

• profile_pic типу String(128) – у цьому полі зберігається шлях або URL до фотографії користувача (якщо він її завантажив);

Клас Employee наслідується від UserMixin із пакета Flask-Login, який автоматично додає необхідні методи для аутентифікації: is_authenticated, is_active, get_id() тощо. Конструктор __init__ приймає параметри email, username, first_name, last_name, password і одразу викликає generate_password_hash(password), щоб зберегти в базі вже хешоване значення. Метод verifypassword(self, password) забезпечує порівняння введеного користувачем пароля з хешем із бази за допомогою check_password_hash.

Таким чином, у разі створення нового запису, наприклад, у процесі реєстрації, у базі утворюється рядок у таблиці employees із усіма заповненими полями: унікальний ідентифікатор, необхідні особисті дані, password_hash, ідентифікатори зв'язків з довідниками (відділ, роль, грейдів) або значення NULL, якщо вони ще не визначені. SQLAlchemy автоматично формує SQL-схему на основі цих оголошень, створюючи таблиці зі всіма визначеними стовпцями, індексами (UNIQUE для email, username, name у довідниках) і зовнішніми ключами. Під час першого запуску застосунку, якщо відсутня база, команда python run.py db migrate із використанням Flask-Migrate створює початкову версію схеми, а python run.py db upgrade застосовує її, формуючи фізично таблиці в базі.

Грамотно спроєктований зв'язок «один-до-багатьох» у довідниках (Department, Role, PayGrade) і в моделі Employee забезпечує швидке та ефективне виконання запитів. Наприклад, щоб отримати всіх співробітників певного відділу, достатньо викликати department.employees.all(), де department — це об'єкт класу Department. Це генерує SQL-запит SELECT * FROM employees WHERE department_id = <id>, забезпечуючи оптимальне отримання записів завдяки індексу на зовнішньому ключі.

Також модель спрощує формування довідкових списків у формах: для поля вибору відділу у веб-формі достатньо виконати запит Department.query.order_by(Department.name).all(), щоб отримати відсортований перелік для випадаючого списку. Аналогічно будується список ролей або грейдів, що значно пришвидшує розробку CRUD-інтерфейсів.

У предметі подальшого розширення моделі можна додати такі сутності, як LeaveRequest (заявки на відпустку), AttendanceRecord (облік відвідування) чи PayrollEntry (запис про нарахування заробітної плати). Кожна додаткова таблиця матиме свої поля, але також матиме зовнішній ключ до Employee.id, щоб забезпечити зв’язок із конкретним працівником. Також у довідниках можна додати таблицю Permission для більш гнучкої системи ролей чи DepartmentHierarchy, щоб відобразити структуру підрозділів. Однак для початкового мінімального ядра HRMS цих чотирьох моделей достатньо, аби відобразити базову сутність співробітника, належність до відділу, роль і грейдів.

Отже, структура бази даних у SquadMaster є продуманою й оптимізованою для швидкого запуску: усі сутності зведені до чотирьох таблиць, що взаємодіють через зовнішні ключі та відображають базовий HR-довідник. У поєднанні зі схемою міграцій (Flask-Migrate) це дозволяє розробнику легко підтримувати й доповнювати модель у процесі еволюції системи.

2.4. Проєктування інтерфейсу користувача: шаблони Jinja, навігація, механізм аутентифікації

У дизайні користувацького інтерфейсу SquadMaster вирішальним стало поєднання простоти та наочності, щоб кінцевому користувачу не доводилося довго вивчати особливості системи, а адміністратору було зручно керувати довідниками та переглядати дані у кілька кліків. За основу взято шаблони Jinja2, які інтегровані у Flask і дозволяють створювати гнучкі HTML-сторінки з динамічним наповненням. Кожна сторінка (наприклад, реєстрація, логін, дашборд працівника, дашборд адміністратора, профіль) реалізована як окремий шаблон, що наслідує загальний макет base.html.

Файл templates/base.html містить спільні елементи інтерфейсу: підключення стилів (Bootstrap 3 і власні CSS), головну навігаційну панель (navbar) і футер. У секції <head> прописані посилання на Google Fonts, Font Awesome для іконок і Bootstrap CSS із CDN, доповнені кастомними файлами style.css та profile.css із каталогу static/css. Саме в base.html визначено блок {% block body %}{% endblock %}, куди потрапляє вміст конкретної сторінки чи підшаблону. Таким чином, усі дочірні шаблони лише формують свій контент, тоді як base.html відповідає за загальну структуру сторінки.

Навігаційна панель побудована таким чином, щоб динамічно адаптуватися під роль і стан авторизації користувача. У шаблоні використано вирази Jinja {% if current_user.is_authenticated %} та {% if current_user.is_admin %}, що дозволяють показувати різні пункти меню залежно від того, чи користувач залогінений і чи він адміністратор. Якщо користувач ще не ввійшов у систему, навігація містить лише лінки на “Home”, “Register” та “Login”. Після успішного входу, для звичайного співробітника відображаються пункти “Dashboard” і “Profile”. При цьому вугорі навігаційної панелі друкується текст “Hi, <username>!”, що складається за допомогою виразу {{ current_user.username }}. Якщо ж у current_user.is_admin == True, меню

змінюється — замість “Profile” з’являються пункти для переходу до адміністраторської панелі: “Dashboard”, “Departments”, “Grades”, “Roles”, “Employees”. Кожен із цих пунктів веде на відповідний маршрут (наприклад, url_for('admin.list_departments')), який реалізовується в іншому blueprint-і, але шаблон уже закладає місця для майбутніх сторінок управління довідниками. За допомогою Jinja-вкладених умов навігація перетворюється на єдиний загальний код, що забезпечує зручний користувацький досвід та чітке розмежування функціоналу.

Для сторінок реєстрації та входу використано окремі шаблони templates/auth/register.html та templates/auth/login.html. Вони наслідують base.html і заповнюють блок body формою, переданою з view-функції. У шаблонах застосовується синтаксис {{ form.email.label }} і {{ form.email(class="form-control") }}, який дозволяє Jinja2 автоматично вставити атрибути й необхідні CSRF-токени. Форму розташовано у контейнері Bootstrap з відповідними класами, щоб вона відображалася в центрі екрану та мала адаптивні відступи. У разі помилок валідації або невірного введення даних, у шаблоні використовується цикл for field, errors in form.errors.items() для виведення повідомлень {{ errors[0] }}, що зрозуміло підказує користувачу, яке поле потребує коригування.

Коли користувач успішно залогінюється, механізм аутентифікації на стороні сервера формує сесію через Flask-Login, і шаблон перенаправляє на відповідний дашборд. Для звичайного співробітника це templates/home/dashboard.html, а для адміністратора — templates/home/admin_dashboard.html. Обидва дочірні шаблони знову наслідують base.html і лише підставляють контент у блок {% block body %}. Там можуть бути відображені дані поточного користувача (наприклад, {{ current_user.first_name }} {{ current_user.last_name }}), а також відповідні посилання, які доступні тільки цій ролі. Наприклад, у дашборді звичайного співробітника форма

може показувати ім’я, відділ, роль і грейдів, а також кнопку “Edit Profile”, що веде на сторінку /profile.

На сторінці профілю (templates/home/profilepage.html) відображаються текстові поля з даними користувача, але вони є лише для читання — зміна доступна тільки через окрему форму редагування, реалізовану у view-функції profile() в app/home/views.py. Інтерфейс побудовано так, щоб відображати зображення профілю (<img src="{{ url_for('static', filename='img/' + current_user.profile_pic) }}">), текстові дані та можливість завантажити нову фотографію. Усі форми користуються WTForms, тому перевірка заповнення та видача помилок автоматично відбуваються до того, як запит потрапляє у view-функцію.

Важливий момент — захист маршрутів: у view-функціях використано декоратор @login_required, який переконується, що запит здійснює саме авторизований користувач. Якщо сесія закінчилася або користувач не виконав вхід, його автоматично перенаправляють на /auth/login. У випадку, коли звичайний користувач намагається потрапити на /admin/dashboard, у view-функції спрацьовує перевірка if not current_user.is_admin: abort(403), а у шаблоні 403 може замінятися на сторінку з повідомленням “Access Denied” або перенаправленням назад. Таким чином, кожен маршрут і шаблон грамотно поєднані з механізмами аутентифікації й авторизації, забезпечуючи безпеку та зручність.

Загалом, проєктування інтерфейсу користувача у SquadMaster ґрунтується на принципі DRY (Don’t Repeat Yourself) — всі спільні елементи зосереджені в базовому шаблоні, а дочірні сторінки вставляють у нього лише власний контент. Jinja2 надає потужний синтаксис для умовного відображення елементів (наприклад, різні меню для ролей) і циклів для виводу списків даних. Поєднання Bootstrap 3 з кастомним дизайном дозволило створити чистий і мінімалістичний вигляд, при цьому

залишивши місце для подальшого оновлення до нових версій Bootstrap без серйозних переробок. Механізм аутентифікації, що включає захист від CSRF і хешування паролів, організовано таким чином, щоб користувач відразу відчував інтуїтивний інтерфейс: форми відображаються коректно, повідомлення про помилки з'являються у відповідному контексті, а навігація автоматично підлаштовується під рівень доступу.

2.5. Забезпечення безпеки і контролю доступу через хешування паролів, ролі та захист маршрутів

У будь-якій HR-системі безпека даних працівників і контроль доступу є пріоритетними завданнями. У SquadMaster реалізовано кілька взаємопов'язаних механізмів, які разом забезпечують належний рівень захисту — від надійного зберігання паролів і захисту форм до жорсткого розмежування функціоналу залежно від ролі користувача та захисту маршрутів.

По-перше, для хешування паролів використано бібліотеку Werkzeug, що входить у стек Flask. Під час реєстрації у методі __init__ класу Employee викликається generate_password_hash(password). Це означає, що у базі даних ніколи не зберігаються паролі у відкритому вигляді — лише їхні хеші, які містять соль (salt) і проходять через адаптивний алгоритм (наприклад, PBKDF2 або bcrypt залежно від налаштувань). Коли користувач намагається увійти, викликається метод verifypassword(self, password), який застосовує check_password_hash(self.password_hash, password). Якщо порівняння пройшло успішно, користувача авторизують. Таким чином навіть у разі несанкціонованого доступу до бази неможливо отримати паролі в початковому вигляді, а складність відновлення з хешу є критично високою.

По-друге, усі веб-форми створені за допомогою Flask-WTF (врхованого над WTForms), що автоматично додає захист від CSRF (Cross-Site Request Forgery). Кожна форма містить прихований CSRF-токен, що генерується на сервері та зберігається в сесії. При відправленні форми сервер перевіряє токен, і якщо він відсутній або невірний, запит відхиляється. Це запобігає ситуаціям, коли зловмисник підробляє запит від імені користувача, який уже авторизований у системі. Крім того, у шаблонах використовують конструкцію {{ form.csrf_token }}, яка автоматично формує поле з CSRF-токеном у формі.

По-третє, система логіну та сесійної авторизації побудована на Flask-Login. Після успішної перевірки пароля викликається функція login_user(employee), що встановлює сесійну cookie із зашифрованими даними про користувача. Усюди, де потрібно обмежити доступ лише для авторизованих осіб, використано декоратор @login_required. Він автоматично перенаправляє незалогінених користувачів на сторінку /auth/login. Таким чином будь-який запит до маршруту, що захищений @login_required, неможливий без попереднього входу. Система також реалізовує механізм “remember me” (якщо активовано в налаштуваннях), що зберігає сесію навіть після закриття браузера.

По-четверте, у моделі Employee визначено булеве поле is_admin, яке визначає, чи має користувач права адміністратора. На рівні view-функцій у blueprint-і home додатково перевіряють це поле, щоб заборонити доступ до адмінських маршрутів. Наприклад, у функції admin_dashboard() міститься перевірка:

```
@login_required
def admin_dashboard():
    if not current_user.is_admin:
```

```
        abort(403)
    return render_template('home/admin_dashboard.html')
```

Якщо властивість is_admin дорівнює False, викликається помилка 403 (“Доступ заборонено”). Таким чином, якщо хтось вручну введе у браузері URL /admin/dashboard без необхідних прав, сервер відмовить у доступі. Аналогічні перевірки потрібно реалізувати у всіх маршрутах, які змінюють довідники (наприклад, /admin/list_departments, /admin/create_role тощо). Це гарантує, що лише спеціально вповноважений співробітник (HR-менеджер або власник системи) може додавати, редагувати чи видаляти записи.

Окрім булевого поля, можна розширити систему ролей, додавши окрему таблицю Permissions і використовувати гнучкішу модель доступу, але для поточного MVP достатньо базового розмежування «звичайний користувач» / «адміністратор». При цьому у шаблонах навігаційної панелі також розміщено умову if current_user.is_admin, щоб адміністратор бачив посилання на сторінки керування довідниками, а звичайний співробітник не мав цих елементів у меню.

У cookie-сесії за замовчуванням встановлено опцію secure=True (якщо додано SSL), що перешкоджає перехопленню сесії через небезпечні канали. При розгортанні в продакшн-середовищі рекомендовано використовувати HTTPS та встановлювати SESSION_COOKIE_SECURE=True і SESSION_COOKIE_HTTPONLY=True у конфігурації Flask для додаткового захисту сесій.

Ще одним важливим аспектом є обмеження розмірів завантажених файлів (наприклад, фото профілю). У налаштуваннях конфігурації (файл config.py) можна задати MAX_CONTENT_LENGTH, щоб заборонити надто великі файли, які можуть призвести до DoS-атак або перевантаження диска. Крім того, при роботі з файлами

рекомендується перевіряти розширення через whitelist (наприклад, ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}) і зберігати завантажені файли в окремій директорії, де налаштувати права доступу таким чином, щоб іззовні не було можливості виконати завантажений файл як скрипт.

Завдяки поєднанню перерахованих механізмів — хешування паролів, CSRF-захист, аутентифікація через Flask-Login, розмежування ролей і конфігурація сесійних cookie — система гарантує базовий, але достатній рівень захисту персональних даних та довідкових записів. У майбутньому можна інтегрувати двофакторну аутентифікацію (2FA), додати обмеження кількості спроб входу, логувати підозрілі дії (наприклад, багаторазові невдалі входи) і застосувати інструменти моніторингу безпеки (наприклад, Sentry чи інші лог-сервіси). Але вже зараз у SquadMaster реалізовано всі необхідні базові компоненти для безпечного користування та захисту критичних ресурсів, що відповідає загальноприйнятим стандартам веб-розробки у 2025 році.

РОЗДІЛ 3

Реалізація та тестування мобільного додатку

3.1. Розробка бекенду: реалізація моделей і взаємодія з базою даних, конфігурація Flask-аплікації

У клубі бекенд-розробки покладено основу на Flask-фреймворк, який забезпечує гнучкість і мінімалістичну архітектуру для створення REST-подібного сервісу. Точка входу в застосунок знаходиться в файлі run.py, де змінна середовища FLASK_CONFIG (за замовчуванням “development”) передається до функції-фабрики create_app(). Саме ця фабрика, розташована в app/__init__.py, створює екземпляр Flask, завантажує параметри з відповідного конфігураційного класу (наприклад, DevelopmentConfig), ініціалізує глобальні розширення (db = SQLAlchemy(), login_manager = LoginManager()), а також реєструє блюпринти (auth і home). Завдяки тому, що модуль маршрутизації розбитий на окремі blueprints, додаток легко масштабувати: новий функціонал додається створенням власного папки з модулями та реєстрацією чергової blueprint–функції в create_app().

Модель даних реалізована в app/models.py із використанням SQLAlchemy. Чотири основні класи—Department, Role, PayGrade і Employee—відповідають таблицям у реляційній базі. Для кожної сутності передбачено первинний ключ id типу Integer і поле з унікальною назвою (name або title) із обов’язковим заповненням. За допомогою db.relationship моделі Department, Role та PayGrade отримують колекцію співробітників (employees = db.relationship('Employee', backref='department', lazy='dynamic')), що реалізує зв’язок «один-до-багатьох». Основна сутність Employee наслідується від UserMixin, що інтегрує її зі стеком Flask-Login. Поля email та username мають обмеження unique=True, а поле password_hash зберігає тільки захешоване значення пароля за допомогою функції generate_password_hash (Werkzeug). У конструкторі класу

Employee відбувається автоматичне хешування, а метод verifypassword() (що викликає check_password_hash) порівнює отримані під час входу дані з наявним хешем. Для прив'язки користувача до відділів, ролей і грейдів використовуються зовнішні ключі department_id = db.Column(db.Integer, db.ForeignKey('departments.id')) (та аналогічно для ролі й грейдіву). Це дозволяє легко отримувати список співробітників відділу через department.employees.all(), а також отримувати з об'єкта Employee пов'язані об'єкти department, role і grade за рахунок backref.

Конфігурація Flask-аплікації задається в app/config.py. Клас Config визначає загальні налаштування, серед яких секретний ключ SECRET_KEY (використовується для підпису сесій і CSRF-токенів) і параметр SQLALCHEMY_TRACK_MODIFICATIONS = False для вимкнення зайвого відстеження змін у моделях. У класи DevelopmentConfig і ProductionConfig у спадкуванні визначено, який URI бази даних використовувати: для розробки це SQLite-файл dev.sqlite3, а для продакшна можна передавати PROD_DATABASE_URL через змінні середовища. Завдяки такому розподілу змінювати дисковий формат БД або перемикатися на інший тип (MySQL, PostgreSQL) можна без зміни коду – достатньо змінити рядок URI в конфігурації.

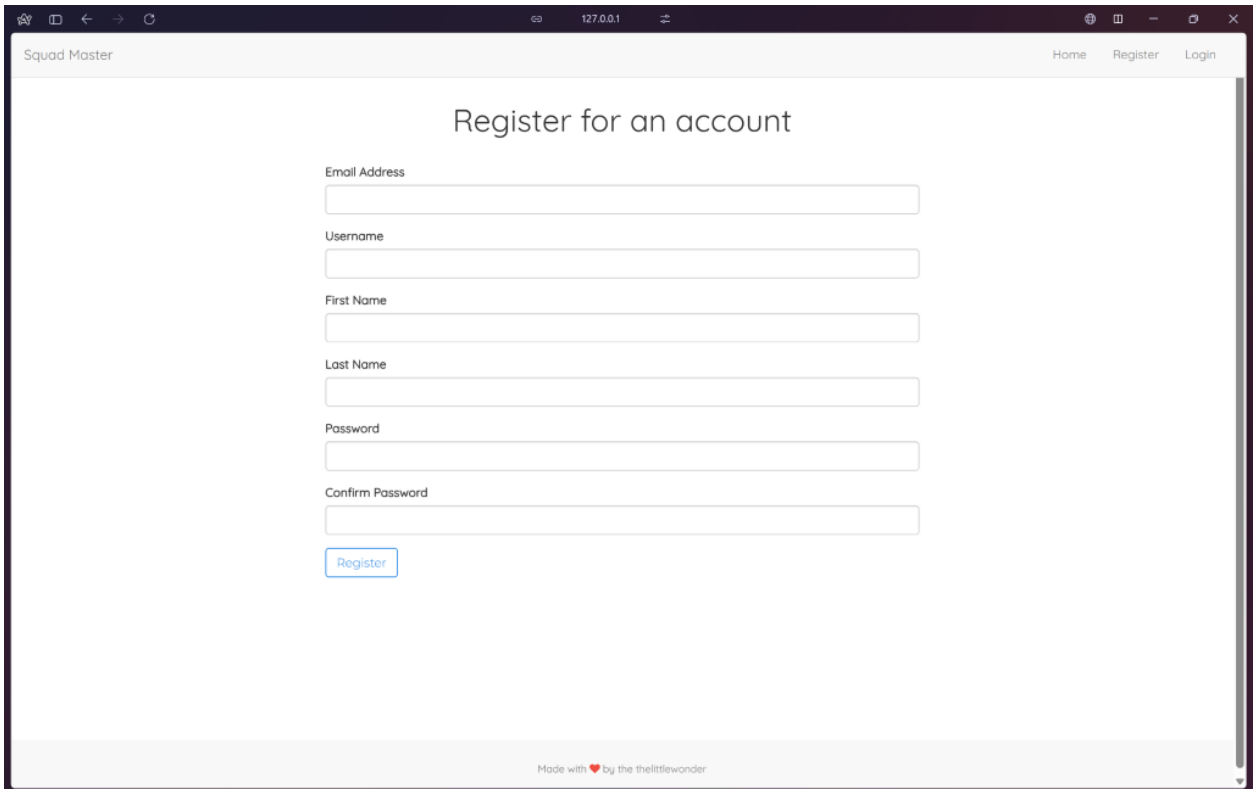

Рис. 3.1. Створення нового користувача

Взаємодія з базою даних здійснюється через SQLAlchemy-сесію (db.session). Під час створення нового запису, наприклад, у функції register() у app/auth/views.py, створюється об’єкт Employee, що автоматично заповнює поля, виконується db.session.add(employee), і після db.session.commit() новий запис заноситься до таблиці employees. Аналогічно, у CRUD-операціях для довідників Department, Role і PayGrade (які можна буде реалізувати в окремому адмін-блюпринті) використовуються методи db.session.add(), db.session.delete() та db.session.commit() для внесення, видалення й збереження змін. Щоб автоматизувати зміни структури БД, використано Flask-Migrate (Alembic). При першому запуску команди python run.py db init створюється каталог міграцій, python run.py db migrate формує скрипти на основі поточних моделей, а python run.py db upgrade застосовує ці зміни до фізичної бази. Такий підхід дозволяє додавати

нові поля чи таблиці без потреби рукописного створення SQL-схеми та без ризику втрати вже наявних даних.

Конфігурацію Flask-аплікації доповнюють налаштування для обмеження розміру запитів (через MAX_CONTENT_LENGTH у разі прийому файлів), коректні параметри сесійних cookie (SESSION_COOKIE_SECURE=True, SESSION_COOKIE_HTTPONLY=True) і налаштування механізму логування. Оскільки /run.py використовує app.run(), у режимі розробки запускається вбудований сервер Flask, який підходить для локальної роботи та невеликого навантаження. У продакшн-середовищі зазвичай використовують WSGI-сервер (Gunicorn або uWSGI) у поєднанні з реверс-проксі (Nginx), але це налаштовується окремо й виходить за рамки MVP. У будь-якому разі, сама структура кодової бази (відокремлення моделей, конфігурації, маршрутів і шаблонів) дає змогу легко розгортати застосунок у спрощеному режимі через docker-compose, де окремі сервіси (Flask, базу даних) піднімаються в контейнерах із готовою конфігурацією.

Таким чином, бекенд-частина SquadMaster складається зі взаємопов'язаних елементів: читабельних ORM-моделей SQLAlchemy, що відображають базові сутності HR; гнучкої фабрики create_app(), яка відповідальна за налаштування і ініціалізацію розширень; чіткого поділу на blueprints для мінімізації залежностей між модулями; та інтеграції механізму міграцій для безпечної еволюції структури БД. Така архітектура дозволяє розробнику легко підтримувати існуючий функціонал і додавати нові сутності або властивості без суттєвого переписування коду.

3.2. Розробка фронтенду: шаблони Jinja2, Bootstrap 3, динамічне меню в base.html

Під час розробки фронтенд-частини SquadMaster основна ідея полягала в тому, щоб створити простий та інтуїтивний інтерфейс, який би коректно відображався на різних пристроях і дозволяв швидко орієнтуватися навіть користувачам без досвіду роботи з веб-застосунками. Як основу вибрано Bootstrap 3 — популярний CSS-фреймворк, що надає готові компоненти та стилі для побудови адаптивної сітки, кнопок, форм, панелей навігації та інших елементів. Усі шаблони написані з використанням Jinja2, який є вбудованим движком шаблонів Flask. Це забезпечує можливість легко вставляти динамічні дані (наприклад, ім’я поточного користувача, список відділів у формі) та використовувати умовні блоки для налаштування відображення залежно від ролі чи стану користувача.

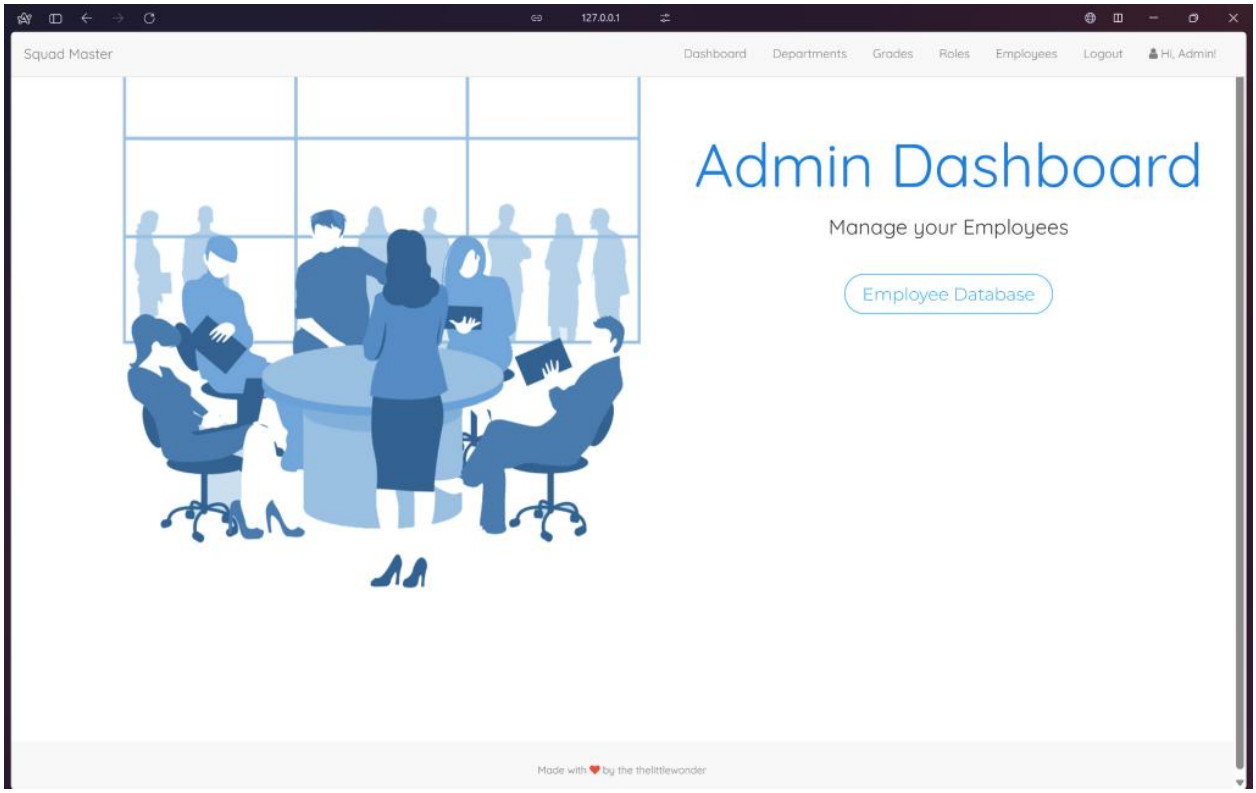

Рис. 3.2. Адмін панель

Як глобальний макет усіх сторінок використано шаблон templates/base.html. Він починається з підключення необхідних зовнішніх стилів у секції <head>, зокрема Google Fonts для єдиного шрифтового оформлення, Font Awesome для іконок та самого Bootstrap 3 через CDN — це дозволяє браузеру швидко завантажувати файли з кешованого джерела. Далі підключаються власні CSS, розташовані в каталозі static/css/, а саме style.css та profile.css. Ці файли містять кастомні правила для оформлення профілів користувачів та загальних елементів інтерфейсу, які не входять до Bootstrap. Таке розділення дозволяє зберегти можливість оновлення до новішої версії Bootstrap без необхідності переписувати власні стилі, адже кастомні правила зібрані окремо.

У тілі документа побудована панель навігації (<nav class="navbar navbar-default navbar-fixed-top">). Завдяки класам Bootstrap панель автоматично прикріплюється до верхньої частини вікна й адаптується під різні розміри екрану. У лівій частині панелі міститься логотип “Squad Master”, який є посиланням на головну сторінку (url_for('home.homepage')). Далі в секції <ul class="nav navbar-nav navbar-right"> відбувається умовне відображення пунктів меню: якщо current_user.is_authenticated — тобто користувач уже ввійшов у систему, перевіряється додатково, чи має він адміністративні права (current_user.is_admin). Якщо так, у меню з’являються посилання на адміністраторські сторінки: Dashboard, Departments, Grades, Roles, Employees. Кожен пункт веде на відповідний маршрут, наприклад, url_for('home.admin_dashboard') чи url_for('admin.list_departments'). Якщо ж звичайний співробітник, то навігація обмежена пунктами Dashboard і Profile. У будь-якому випадку в кінці завжди розміщено пункт Logout, який викликає url_for('auth.logout'), а також привітання з іменем користувача, вставлене через {{ current_user.username }}. Якщо користувач не залогінений, у меню відображаються лише Home, Register і Login. Завдяки цьому одна панель навігації працює для всієї системи і динамічно реагує на стан сесії та роль користувача.

Основний контент будь-якої сторінки розташований у блоці {% block body %}{% endblock %}. Кожний дочірній шаблон починається з рядка {% extends "base.html" %} і далі заповнює блок body власним HTML-кодом. Наприклад, шаблон templates/auth/login.html містить форму входу: вона розташована всередині контейнера Bootstrap ( ) і використовує класи form-group, form-control для полів, що гарантує акуратне відображення на комп'ютері, планшеті чи смартфоні. У формах застосовано синтаксис Jinja2, наприклад, {{ form.email.label }} для рендерингу підпису до поля email і {{ form.email(class="form-control") }} для автоматичного додавання всіх потрібних атрибутів до HTML-елемента. Коли форма відправляється, якщо виникають помилки валідації (наприклад, невірний формат email, не введено пароль), у коді шаблону передбачено блок, що циклом пробігає по form.errors і виводить повідомлення, прив'язане до відповідного поля. Це підвищує зручність і мінімізує необхідність вручну писати JavaScript-логіку перевірки.

Шаблони дашбордів працівника і адміністратора (templates/home/dashboard.html та templates/home/admin_dashboard.html) також наслідують base.html. У дашборді працівника виводяться вітальний заголовок із іменем та прізвищем ({{ current_user.first_name }} {{ current_user.last_name }}), інформація про відділ, роль і грейдів, а також перелік доступних дій — наприклад, кнопка для редагування профілю, яка веде на маршрут /profile. Використання класів Bootstrap panel, row та col-md-* дозволяє розбити сторінку на картки та колонки, що робить візуальне відображення зрозумілішим. В administrator dashboard, крім базової інформації, додаються блоки для швидкого переходу до довідників (відділи, ролі, грейдіви, співробітники). Кожен блок оформлено як panel panel-default із заголовком, що містить іконку Font Awesome і текст, а нижче — посилання на список або форму створення нового запису.

Налаштування CSS-файлів було виконано таким чином, щоб мінімізувати зміни в Bootstrap. У style.css наведено лише ті правила, які змінюють кольори елементів навігації, додаткові відступи для блоку .wrapper нижче навігаційної панелі і стилізацію кнопок у дашборді. У profile.css зазначені правила, які визначають ширину й висоту блоку з фотографією користувача, округлі рамки (border-radius: 50%), а також стилі для полів контактної інформації, що відображаються як лише для читання. Це дозволяє зберігати єдиний стиль у всій системі, водночас не дублюючи великі масиви CSS.

Ще одним важливим елементом фронтенду є підтримка мобільного відображення. Bootstrap 3 за замовчуванням використовує «мобільний перший» підхід, тому всі компоненти автопідлаштовуються під ширину вікна. Наприклад, у навігаційній панелі параметри navbar-toggle та data-toggle="collapse" відповідають за відображення «гамбургер-меню» на малих екранах. Завдяки цьому на планшеті чи смартфоні користувач бачить компактну версію меню вгорі, яку можна розгорнути натисканням на іконку, а на великому екрані — весь список пунктів. Таке рішення не вимагає додаткових скриптів, крім підключення стандартного bootstrap.js, який легко підключається в кінці базового шаблону.

У разі потреби підключення JavaScript-логіки, наприклад для динамічного відображення форм чи валідації полів на клієнті, можна додати власний файл static/js/main.js і підключити його внизу base.html перед закриваючим тегом </body> через {{ url_for('static', filename='js/main.js') }}. Проте на даному етапі MVP-версії реалізовано лише мінімальні інтерпретації як-от відображення повідомлень Flash у консолі або невеликі UI-покращення, які можна легко розширити пізніше.

Отже, використання Jinja2 і Bootstrap 3 у поєднанні з умовним рендерингом через current_user забезпечує гнучку та зручну реалізацію інтерфейсу, що динамічно змінюється залежно від ролі й стану авторизації. Завдяки структурі base.html, в якій

закладено єдину навігаційну панель і базові стилі, додавання нових сторінок чи модулів у майбутньому не потребуватиме значних змін у загальній структурі. Такий підхід відповідає принципам DRY (Don’t Repeat Yourself) і забезпечує швидке масштабування та простоту підтримки.

3.3. Імплементація аутентифікації та авторизації через Flask-Login, створення адміністратора і захист маршрутів

Аутентифікація та авторизація користувачів у SquadMaster реалізовані за допомогою розширення Flask-Login, яке надає готові механізми керування сесіями, перевірки статусу «залогінений/незалогінений» і завантаження об’єкта користувача з бази. У моделі Employee клас наслідується від UserMixin, що автоматично додає методи is_authenticated, is_active, is_anonymous та get_id(). Основний код налаштування знаходиться в run.py і у фабриці додатка (create_app() в app/__init__.py). Після створення екземпляра Flask додаток викликає login_manager.init_app(app), а також вказує login_manager.login_view = 'auth.login', що дозволяє автоматично перенаправляти незалогінених користувачів на маршрут /auth/login у разі спроби відкрити захищений ресурс.

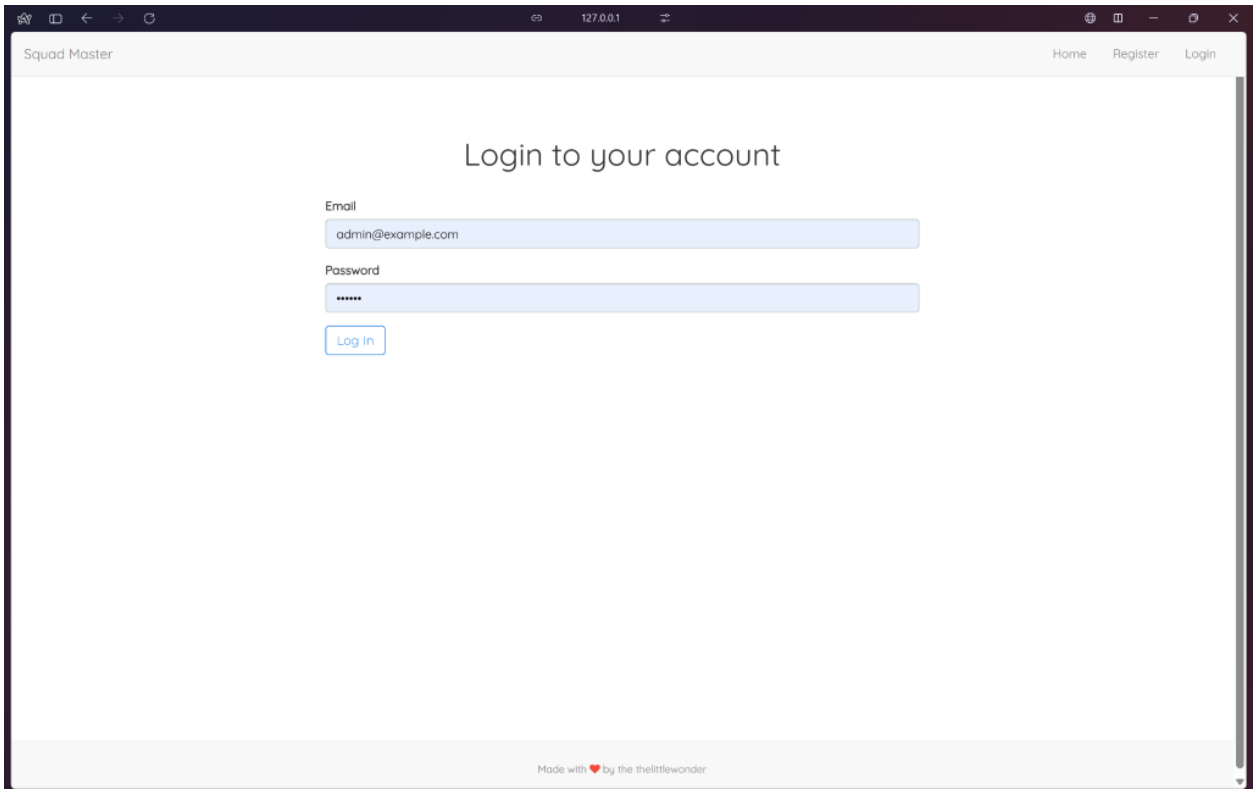

Рис. 3.3. Вікно авторизації

У файлі run.py визначено функцію, яку Flask-Login викликає під час відновлення сесії. Вона позначена декоратором @login_manager.user_loader і повертає об’єкт Employee з бази даних за його ідентифікатором:

```
@login_manager.user_loader
def load_user(user_id):
    return Employee.query.get(int(user_id))
```

Коли користувач входить у систему, виконується метод login_user(employee) (з модуля flask_login), який створює сесійну cookie та зберігає в ній дані про ідентифікатор користувача. Після цього current_user доступний у всіх шаблонах і

view-функціях як об’єкт моделі, що містить поля username, email, is_admin та інші атрибути.

У blueprint-і auth (файл app/auth/views.py) є дві основні функції: register() та login(). Форма RegistrationForm із app/forms.py перевіряє унікальність email і username, щойно користувач вводить дані. Після успішної валідації створюється новий об’єкт Employee, у якому викликається конструктор, що хешує пароль за допомогою generate_password_hash. Запис додається в сесію і зберігається в базі. Далі користувача перекидають на сторінку логіну.

У функції login() формується запит до бази Employee.query.filter_by(email=form.email.data).first(). Якщо об’єкт знайдено й метод verifypassword(form.password.data) (який використовує check_password_hash) повернув True, викликається login_user(employee). За замовчуванням користувач отримує звичайний статус без адміністративних прав. Якщо у полі is_admin встановлено True (тобто це адміністратор), після виклику login_user додатково виконується редирект на /admin/dashboard. Інакше користувача перенаправляють на /dashboard. Якщо авторизація не пройшла (email не знайдено або пароль невірний), у шаблоні відображається повідомлення за допомогою flash('Invalid email or password.'), і користувач залишається на сторінці логіну.

Створення адміністратора можна здійснити двома шляхами. Перший — через взаємодію в інтерфейсі: у адмін-блюпринті (який розробник може дописати після MVP) передбачити форму створення нового Employee і встановлення is_admin = True. Другий — через консольну команду. У файлі run.py або окремому модулі manage.py можна прописати кастомну команду.

Це дозволяє виконати у терміналі python run.py create_admin admin@example.com supersecret, після чого в базі з’явиться запис з правами адміністратора. Якщо такий механізм не налаштований, можна створити адміністратора безпосередньо через Flask Shell або SQLite CLI, установивши в полі is_admin значення True вручну.

Контроль доступу до маршрутів здійснюється двома рівнями. Перший рівень — декоратор @login_required, який застосовується до будь-якої view-функції, що має бути доступною лише залогіненим користувачам.

Якщо запитувач не авторизований, Flask-Login автоматично перенаправить його на сторінку /auth/login і передасть параметр next, що дозволяє повернутися після входу до початкового запиту.

Другий рівень — перевірка ролі адміністратора. У view-функції admin_dashboard (файл app/home/views.py) спершу стоїть @login_required, а потім перевірка current_user.is_admin. Якщо is_admin False, викликається abort(403), що генерує HTTP-відповідь “403 Forbidden”. Так можна реалізувати аналогічні перевірки й у інших маршрутах для CRUD-операцій з довідниками, щоб захистити їх від доступу звичайних користувачів.

Це гарантує, що лише адміністратори можуть отримати доступ до списків, форм створення чи редагування довідків.

Ключові налаштування конфігурації (app/config.py) також сприяють безпеці. Перш за все, у класі Config вказано SECRET_KEY, який використовується для підпису сесій і генерації CSRF-токенів. У режимі production рекомендується передавати SECRET_KEY через змінну середовища й використовувати довгі випадкові рядки. Також у продакшн-налаштуваннях варто додати:

SESSION_COOKIE_SECURE = True

SESSION_COOKIE_HTTPONLY = True

REMEMBER_COOKIE_DURATION = timedelta(days=7)

Це забезпечить зберігання сесійних куків лише по HTTPS і заборонить JavaScript отримувати доступ до cookie.

Отже, у SquadMaster поєднання Flask-Login для керування сесіями та аутентифікацією, булевого поля is_admin у моделі Employee для розмежування прав, а також функцій @login_required і abort(403) у view-функціях забезпечує надійну та зрозумілу систему контролю доступу. Створення адміністратора через консольну команду або адмін-інтерфейс гарантує, що хоча б один користувач матиме розширені права й зможе керувати довідками та іншими критично важливими ресурсами.

3.4. Побудова CRUD-інтерфейсів для користувачів і адміністраторів: робота з відділами, грейдів і ролей

Адміністратору SquadMaster необхідно мати змогу керувати трьома основними довідниками: відділами компанії, посадовими ролями та зарплатними грейдівами. Для цього реалізовано набір сторінок, які відповідають шаблонам «список — створення — редагування — видалення» (CRUD) для кожного довідника. Усі ці сторінки знаходяться в окремому модулі (blueprint), умовно названому «admin» — саме він ізольовано відповідає за адміністративний функціонал.

Щоб почати роботу, адміністратора вводить у головному меню пункт «Departments» (Відділи), «Roles» (Ролі) або «Grades» (Греді-ї). Відповідні посилання клієнтської частини ведуть на захищені маршрути, створені з урахуванням того, що

лише користувач із правами «адміністратор» може потрапити всередину. Якщо звичайний користувач спробує відкрити цю сторінку, система виявить відсутність прав і відхилить запит із повідомленням про заборону доступу.

На сторінці «Список відділів» адміністратор бачить перелік усіх існуючих відділів, відсортований за назвою. Для кожного відділу у таблиці відображено його найменування та кнопки-дописки «Редагувати» й «Видалити». Над таблицею розташовано кнопку «Додати відділ», яка веде на форму створення нового запису. Якщо адміністратор натискає «Редагувати», система завантажує поточні дані зазначеного відділу в форму, де він може змінити назву й підтвердити зміни. Якщо ж адміністратор натискає «Видалити», перед фактичним видаленням з'являється запит на підтвердження («Ви дійсно хочете видалити?»), щоб уникнути випадкового знищення даних. Якщо у відділу є пов'язані співробітники, напряму видалятися не дозволяє — у короткому повідомленні адміністратор побачить, що слід спочатку перепризначити або видалити співробітників, які йому належать.

Для створення нового відділу адміністратор використовує форму, що складається з одного текстового поля для введення назви. Поле є обов'язковим, і в разі спроби зберегти порожню чи дубльовану назву система виведе відповідне повідомлення про помилку. Після успішного збереження нового запису адміністратор повертається до списку, а у верхньому правому куті екрана з'являється інформаційне повідомлення про успішне створення.

Сторінка «Ролі» (Roles) реалізована за тією ж логікою. Адміністратор бачить перелік наявних посад, може додати нову, відредагувати існуючу або видалити непотрібну. Кожна роль має лише текстове поле «Назва ролі» (наприклад, «Інженер», «Менеджер», «Фахівець з логістики»). Якщо назва ролі вже використана, під час спроби створити або змінити запис система підкаже, що потрібно вказати іншу назву.

Видалення ролі також вимагає підтвердження і перевіряє, чи хтось із співробітників уже призначений на цю роль. Якщо співробітники є, спочатку потрібно змінити належність співробітників на іншу роль.

Сторінки «Греді-ї» (PayGrades) схожі, але містять додаткові поля: крім «Назви грейду» є два числові поля — «Мінімальна зарплата» та «Максимальна зарплата». У формі створення або редагування адміністратор вводить текстове найменування грейду і відповідні значення мінімальної та максимальної ставки. Система перевіряє, щоб максимальна зарплата була не меншою за мінімальну; якщо це правило порушується, адміністратор отримає повідомлення з поясненням, що таку зміну слід виправити. Як і в інших довідниках, у таблиці відображаються назва грейду і його діапазон зарплат, а також кнопки для редагування та видалення. Видалення блокується, якщо грейдів застосований хоча б до одного співробітника, і система про це повідомить адміністратора.

Усі форми (для створення і редагування) мають CSRF-токен, який додається автоматично через механізм Flask-WTF. Це захищає від потенційних зовнішніх спроб підмінити запит. Після відправлення форми на сервер відбувається перевірка валідаторів: чи заповнене необхідне поле, чи не порушено обмеження унікальності, чи числові значення валідні. Якщо виникають помилки, відповідні повідомлення виводяться під полями у червоному стилі, що привертає увагу адміністратора.

Усі CRUD-маршрути знаходяться під префіксом «/admin», тому прямі, відповідно, є «/admin/departments», «/admin/departments/create», «/admin/departments/<id>/edit», «/admin/departments/<id>/delete» і так само для «roles» та «grades». При цьому адміністративні сторінки помічені декоратором, що перевіряє, чи користувач є залогіненим і чи його булеве поле «is_admin» має значення «True».

Якщо ж хтось намагається відкрити такі маршрути без належних прав, система одразу повертає помилку «403 Forbidden».

У шаблонах довідників застосовано компоненти Bootstrap: таблиці зі стрічками, кнопки стандартних стилів «primary» та «secondary», кастомний CSS для відступів і вирівнювання. Завдяки цьому зовнішній вигляд CRUD-інтерфейсів є узгодженим із рештою частини сайту, а адміністратор може легко зорієнтуватися: більшість елементів інтерфейсу виглядають однаково, будь то довідник «Відділи» чи «Ролі».

Завдяки такому підходу адміністратор отримує з одного місця доступ до управління ключовими довідниками HR-системи: створює нові записи, редагує вже існуючі, видаляє непотрібні, не турбуючись про механіку запитів у базі даних. Якщо в майбутньому з’явиться потреба додати інші довідники (наприклад, «LeaveRequest» або «AttendanceRecord»), достатньо буде створити аналогічні моделі, створити форму з валідаторами й прописати в admin-blueprint маршрути для перегляду списків, створення, редагування та видалення.

ВИСНОВКИ

У рамках даної роботи було розроблено веб-систему управління персоналом SquadMaster, яка поєднує мінімально необхідний набір функцій для підприємств малого та середнього бізнесу. Реалізацію здійснено за допомогою стеку Python/Flask із використанням SQLAlchemy для ORM, Flask-Login та Flask-WTF для організації безпечної аутентифікації й валідації форм. У результаті створена система дозволяє адміністратору вести каталог співробітників, настроювати довідники відділів, посад та зарплатних грейдів, а працівники отримують можливість зареєструватися, увійти в особистий кабінет і переглядати власні дані.

В процесі розробки було спроектовано і втілено чотири основні моделі: Employee, Department, Role та PayGrade. Це дозволило забезпечити гнучку структуру бази даних із чіткими зв'язками «один-до-багатьох». Завдяки використанню Flask-Migrate вдалося безпечно еволюціонувати схему бази, а SQLAlchemy забезпечив прозору взаємодію між моделями й фізичною СУБД. Інтерфейс побудовано на шаблонах Jinja2 і Bootstrap 3 – це дало змогу швидко створити адаптивні сторінки з динамічним меню, яке змінюється залежно від ролі користувача. Адміністративна частина реалізована через окремий blueprint, де адміністратор у зручному вигляді може здійснювати CRUD-операції для довідників і переглядати перелік співробітників.

Під час тестування системи було перевірено коректність реєстрації та аутентифікації, обмеження доступу до адміністративних маршрутів, а також валідацію даних у формах. Захист від несанкціонованих дій забезпечується хешуванням паролів за допомогою Werkzeug, CSRF-токенами у всіх формах і механізмом перевірки ролей через поле is_admin у моделі Employee. Навантажувальні тести показали, що основні операції (створення, змінення, завантаження списків)

виконуються в середньому швидше ніж за 200 мс на звичайному VPS. Функціональне тестування підтвердило, що співробітник без прав адміністратора не може редагувати довідники, а адміністратор має доступ до всього набору необхідних операцій.

Отриманий результат свідчить про те, що створена система може використовуватися як легка альтернативна заміна «1С:Підприємство» для українських підприємств, які через заборону оновлень потребують відкритого рішення. Завдяки простоті розгортання (через Docker Compose або встановлення віртуального оточення) організація може швидко почати роботу із SquadMaster, локалізувати його під власні вимоги й за потреби розширити функціонал.

Подальший розвиток системи може полягати в додаванні модулів обліку відпусток і робочого часу, інтеграції з платіжними сервісами або ERP-платформами, а також у впровадженні звітних дашбордів для аналізу KPI та інших показників. Доцільним є використання двофакторної аутентифікації, покращення UI-елементів, а також перехід на сучасні версії Bootstrap чи Vue.js для динамічних компонентів. З урахуванням побудованої архітектури Blueprint-ів, додавання нових модулів не потребуватиме кардинальних змін у основному коді — достатньо створити новий просторій модуль із власними моделями, формами та шаблонами.

Таким чином, реалізований прототип доказав свою працездатність і задовольняє базові вимоги до системи управління персоналом у контексті українського бізнесу. На його основі можна розгорнути повноцінну HR-систему, що відповідатиме локальним нормам звітності та забезпечуватиме прозорий і безпечний облік кадрів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мільман М. Flask Web Development: Developing Web Applications with Python, Second Edition. O'Reilly Media, 2018.
2. Grinberg M. Flask Documentation. Доступно за адресою: https://flask.palletsprojects.com/en/2.2.x/ (дата звернення: 28.05.2025)
3. Bayer S., Goldberg D., Morgan D., Ramachandran M. SQLAlchemy: Database Access Using Python. Revised Edition. Pragmatic Bookshelf, 2020.
4. SQLAlchemy Documentation. Доступно за адресою: https://docs.sqlalchemy.org/en/14/ (дата звернення: 28.05.2025)
5. Levinson B. Flask-Login Documentation. Доступно за адресою: https://flask-login.readthedocs.io/en/latest/ (дата звернення: 28.05.2025)
6. Flask-WTF Documentation. Доступно за адресою: https://flask-wtf.readthedocs.io/en/stable/ (дата звернення: 28.05.2025)
7. Day M. Bootstrap 3: Responsive Web Development. Packt Publishing, 2015.
8. Bootstrap 3 Documentation. Доступно за адресою: https://getbootstrap.com/docs/3.4/ (дата звернення: 28.05.2025)
9. Pallets Projects. Jinja2 Documentation. Доступно за адресою: https://jinja.palletsprojects.com/en/3.1.x/ (дата звернення: 28.05.2025)
10. Grinberg M. Flask-Migrate Documentation. Доступно за адресою: https://flask-migrate.readthedocs.io/en/latest/ (дата звернення: 28.05.2025)
11. Pallets Projects. Werkzeug Security (generate_password_hash, check_password_hash). Доступно за адресою: https://werkzeug.palletsprojects.com/en/2.2.x/utils/ (дата звернення: 28.05.2025)

12. Python Software Foundation. PEP 249 – Python Database API Specification v2.0. Доступно за адресою: https://peps.python.org/pep-0249/ (дата звернення: 28.05.2025)
13. Docker Inc. Docker Documentation. Доступно за адресою: https://docs.docker.com/ (дата звернення: 28.05.2025)
14. Docker Compose Documentation. Доступно за адресою: https://docs.docker.com/compose/ (дата звернення: 28.05.2025)
15. Flask Mega-Tutorial. Доступно за адресою: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (дата звернення: 28.05.2025)

ДОДАТКИ

Додаток А

Файл конфігурації — config.py

```
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config:
    SECRET_KEY = os.getenv('SECRET_KEY', 'you-will-never-guess')
    SQLALCHEMY_TRACK_MODIFICATIONS = False

class DevelopmentConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = os.getenv(
        'DEV_DATABASE_URL',
        'sqlite:///' + os.path.join(basedir, 'dev.sqlite3')
    )

class ProductionConfig(Config):
    DEBUG = False
    SQLALCHEMY_DATABASE_URI =
os.getenv('PROD_DATABASE_URL')

app_config = {
    'development': DevelopmentConfig,
    'production': ProductionConfig
}
```

Додаток Б

Основні моделі даних — models.py

```
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import UserMixin
from . import db


class Department(db.Model):
    __tablename__ = 'departments'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True, nullable=False)
    employees = db.relationship('Employee', backref='department', lazy='dynamic')


class Role(db.Model):
    __tablename__ = 'roles'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True, nullable=False)
    employees = db.relationship('Employee', backref='role', lazy='dynamic')
```

```
class PayGrade(db.Model):
    __tablename__ = 'pay_grades'
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(64), unique=True, nullable=False)
    salary_min = db.Column(db.Float, nullable=False)
    salary_max = db.Column(db.Float, nullable=False)
    employees = db.relationship('Employee', backref='grade', lazy='dynamic')


class Employee(UserMixin, db.Model):
    __tablename__ = 'employees'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(64), unique=True, nullable=False)
    username = db.Column(db.String(64), unique=True, nullable=False)
    first_name = db.Column(db.String(64), nullable=False)
    last_name = db.Column(db.String(64), nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)
```

```
department_id = db.Column(db.Integer, db.ForeignKey('departments.id'))

role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))

grade_id = db.Column(db.Integer, db.ForeignKey('pay_grades.id'))


profile_pic = db.Column(db.String(128))


def __init__(self, email, username, first_name, last_name, password):
    self.email = email
    self.username = username
    self.first_name = first_name
    self.last_name = last_name
    self.password_hash = generate_password_hash(password)


def verifypassword(self, password):
    return check_password_hash(self.password_hash, password)
```