

# ГРАДЬНА КОМОРДА (НЕ PROGRAMISTY)

Сучасний E-commerce вебзастосунок для продажу настільних ігор

Команда: Ne\_Programisty

Дата здачі: 16.12.2025

# СКЛАД КОМАНДИ

## 🚀 Команда «Ne\_Programisty»

-  **Лукащук Данило** — *Team Lead / Fullstack*  
Архітектура проекту, Docker & Nginx, Адмін-панель.
-  **Чорноус Сергій** — *Frontend Dev*  
Дизайн інтерфейсу, Кошик, Логіка замовень.
-  **Базюк Максим** — *Backend Dev*  
Система відгуків (Feedback), Новини, анімації.
-  **Кондратюк Дмитро** — *QA / UI/UX Designer*  
Тестування (Bug hunting), Адаптивність.



# ОПЛЯД ПРОБЛЕМІ

**Проблема:** Ринок настільних ігор зростає, але користувачам часто не вистачає зручних, швидких та сучасних платформ для їх придбання. Багато існуючих сайтів мають застарілий дизайн, складну навігацію або відсутність мобільної адаптації.

## Цільова аудиторія:

- Фанати настільних ігор (гіки).
- Компанії друзів, що шукають розваги.
- Батьки, що шукають розвиваючі ігри для дітей.

**Актуальність:** Створення "лампового", але технологічного простору для ком'юніті з можливістю швидкої покупки, читання новин та обміну відгуками.



# РІШЕННЯ

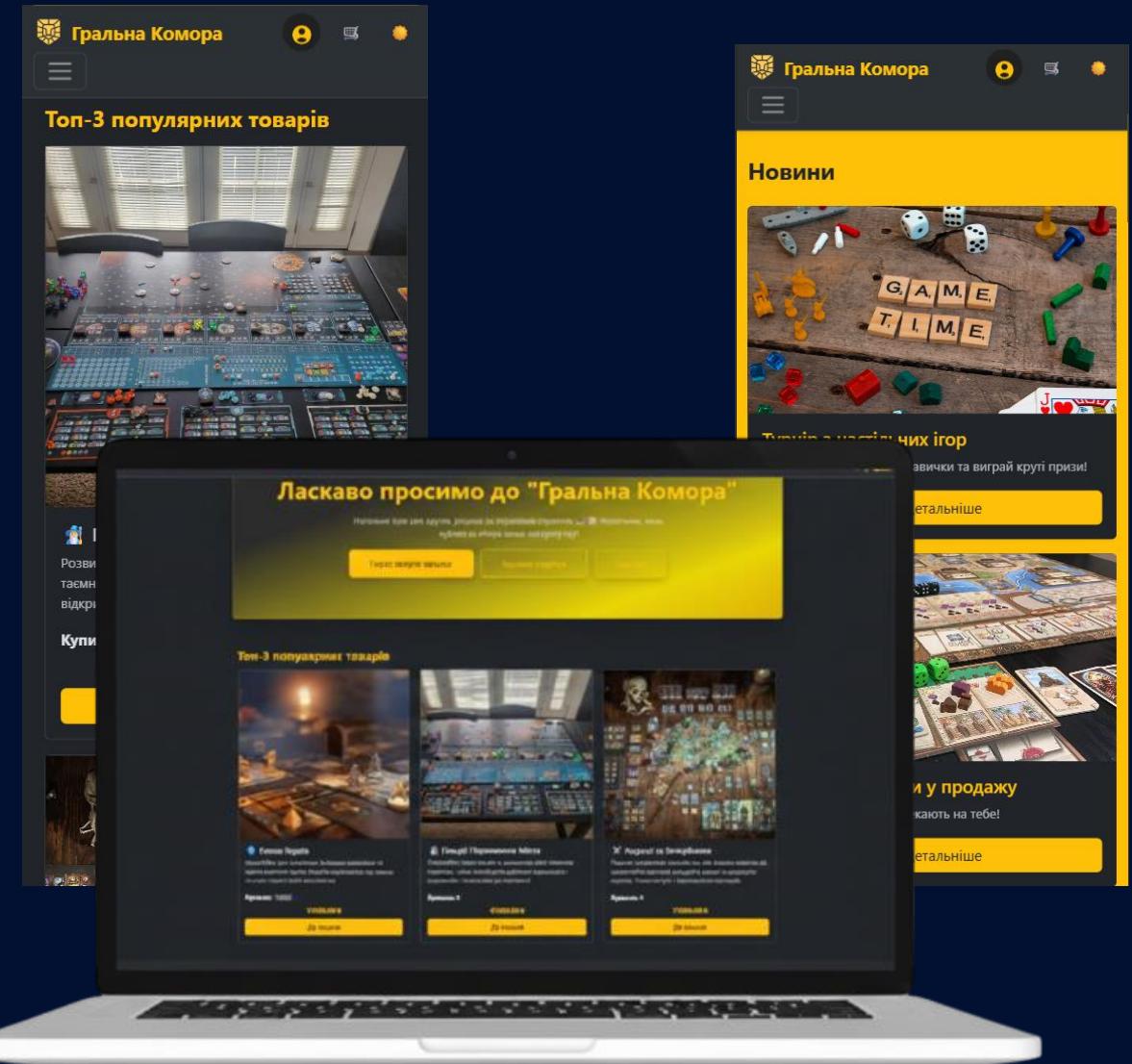
Наше рішення: "Гральна Комора" – це повнофункціональний вебсервіс, що поєднує в собі каталог товарів, CMS (систему управління контентом) та RESTful API.

## Ключова цінність:

- ⚡ Швидкість: Оптимізований бекенд та легкий фронтенд.
- 📱 Зручність: Адаптивний дизайн (Mobile-First) та інтуїтивний UX.
- 🛡️ Надійність: Безпечна авторизація та стабільна робота завдяки контейнеризації.

## Унікальні особливості:

- Власні CSS-анімації та лоадер.
- Темна/Світла тема.
- Гнучка система ролей (User, Moderator, Admin).



# ТЕХНОЛОГІЧНИЙ СТЕК

Ми обрали сучасний стек для балансу між швидкістю розробки та продуктивністю.

## Frontend:

- 🎨 HTML5 / CSS3 (Custom Animations)
- 💎 Bootstrap 5 (Адаптивність)
- ⚡ JavaScript (ES6+) (Fetch API, динаміка)

## Backend:

- 🐍 Python 3.13
- 🚀 Flask 3.0 (Web Framework)
- 🗁 SQLAlchemy + Alembic (ORM & Migrations)
- 🔒 Flask-JWT-Extended (Security)

## Інфраструктура:

- 🚒 Docker & Docker Compose (Containerization)
- 🐄 Nginx (Reverse Proxy)
- 🐳 Git & GitHub (Version Control)



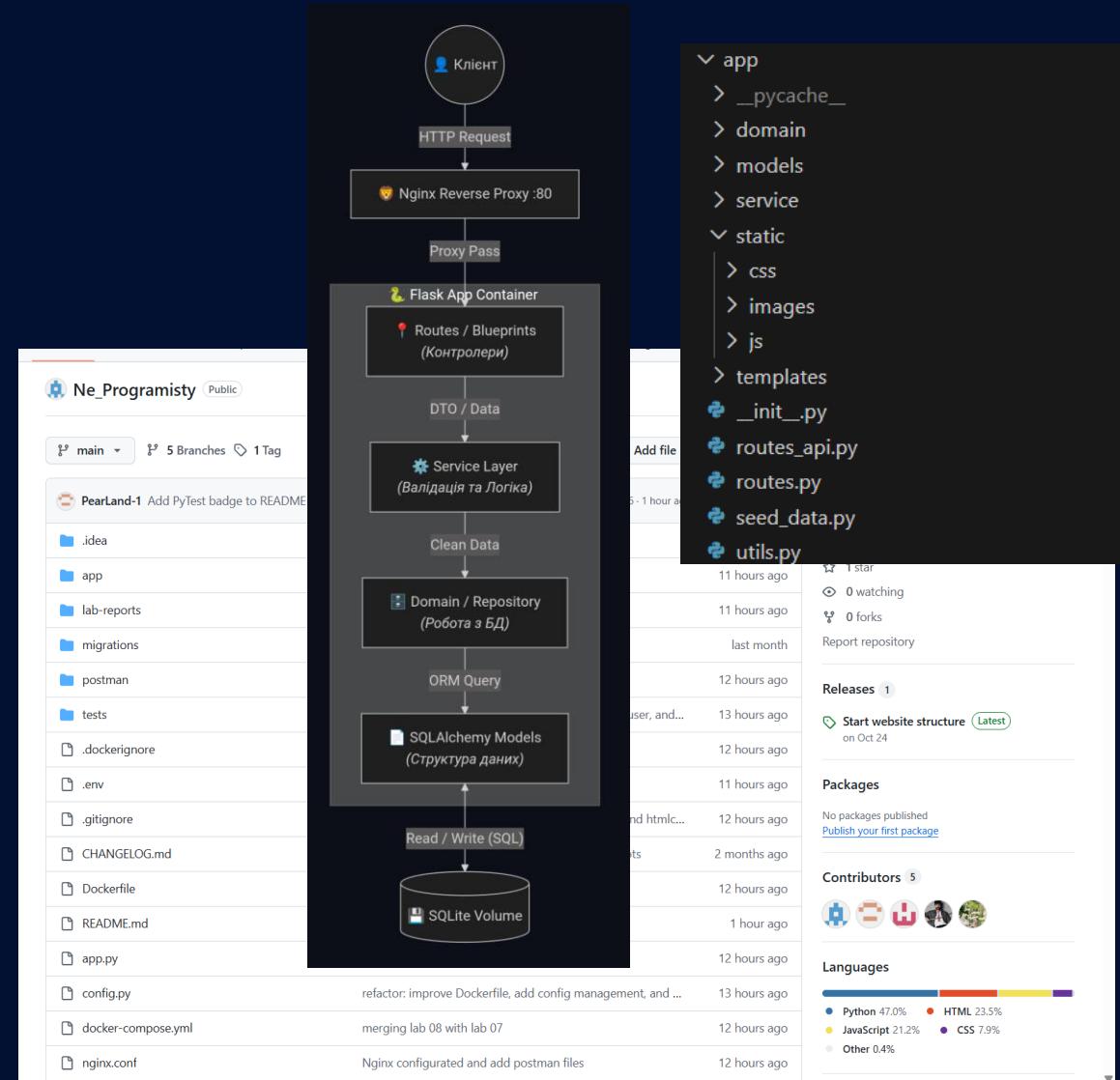
# АРХІТЕКТУРА СИСТЕМІ

Ми реалізували **Layered Architecture** (Багатошарову архітектуру) для чистоти коду.

**Внутрішня структура Flask:**

1. **Routes:** Приймають запити (Controllers).
2. **Service Layer:** Бізнес-логіка та валідація даних.
3. **Domain/Repository:** Пряма робота з БД.
4. **Models:** Опис таблиць SQLAlchemy.
5. **Database:** SQLite Volume.

**Взаємодія:** Nginx приймає запити, передає їх у Flask, де вони проходять через шари валідації та бізнес-логіки перед записом у БД.



# ОСНОВНИЙ ФУНКЦІОНАЛ

## 🔒 Аутентифікація:

- Реєстрація та вхід (JWT Access/Refresh токени).
- Відновлення пароля через email.

The top window shows a modal titled 'Available authorizations' with a 'Bearer (apiKey)' section. It describes a JWT Authorization header using the Bearer scheme, with fields for Name: 'Authorization', In: 'header', and Value: a placeholder input field. A green 'Authorize' button is at the bottom right. The bottom window shows a dark-themed password reset form with fields for 'Текущий пароль' (Current password), 'Новий пароль' (New password), 'Підтвердіть пароль' (Confirm password), and a 'Змінити пароль' (Change password) button with a back arrow.

The screenshot shows a catalog page for board games. It features three main items: 'STAR SAGA' (35 000 ₴), 'Колоніатори Міфічних Земель' (15 000 ₴), and 'Королі та Загарники' (25 000 ₴). Each item has a thumbnail, a brief description, and a 'Добавити в кошик' (Add to cart) button. The page includes a navigation bar with links like 'Головна', 'Про нас', 'Каталог', 'Контакти', 'Новини', 'Відгуки', 'адмін панель', and a search bar.

## 🛍️ E-commerce:

- Каталог товарів з фото та описом.
- Кошик (додавання, зміна кількості, видалення).
- Оформлення замовлення (Checkout).

# ОСНОВНИЙ ФУНКЦІОНАЛ

## ⚙️ Адмін-панель:

- CRUD для товарів, новин, користувачів.
- Модерація відгуків.
- Керування статусами замовлень.

The Admin Panel interface is shown in two screenshots. The top screenshot shows the 'News' section with a list of news items and a 'Add news' button. The bottom screenshot shows the 'Orders' section with a table listing orders by ID, customer, status, and amount, along with buttons to manage each order.

## Новини



### Новинка: Гра "Стратегія 2025"

Випробуйте свої стратегічні навички у новій грі! "Відкрийте для себе світ бітв і дипломатії.

[Детальніше](#)



### Акція: -40% на популярні ігри

Обмежений час! Знижки на топові настільні ігри цього тижня — поповніть колекцію за вигідною ціною!

[Детальніше](#)



### Майстер-клас для гравців

Хочеш грати як професіонал? Приходь на наш безкоштовний майстер-клас і навччись новим тактикам!

[Детальніше](#)

## 📰 Контент:

- Стрічка новин.
- Система користувачів.

відгуків

# НОВІ ФУНКЦІЇ (LAB 3)

В рамках останнього етапу ми впровадили Контейнеризацію.

Що додано:

- 🐳 Docker Support: Створено оптимізований Dockerfile на базі `python:3.11-alpine`.
- 🐋 Orchestration: Налаштовано `docker-compose` для зв'язку сервісів `web` та `nginx`.
- nginx Configuration: Налаштовано проксі-сервер для роздачі статики та захисту застосунку.

Як це покращує застосунок:

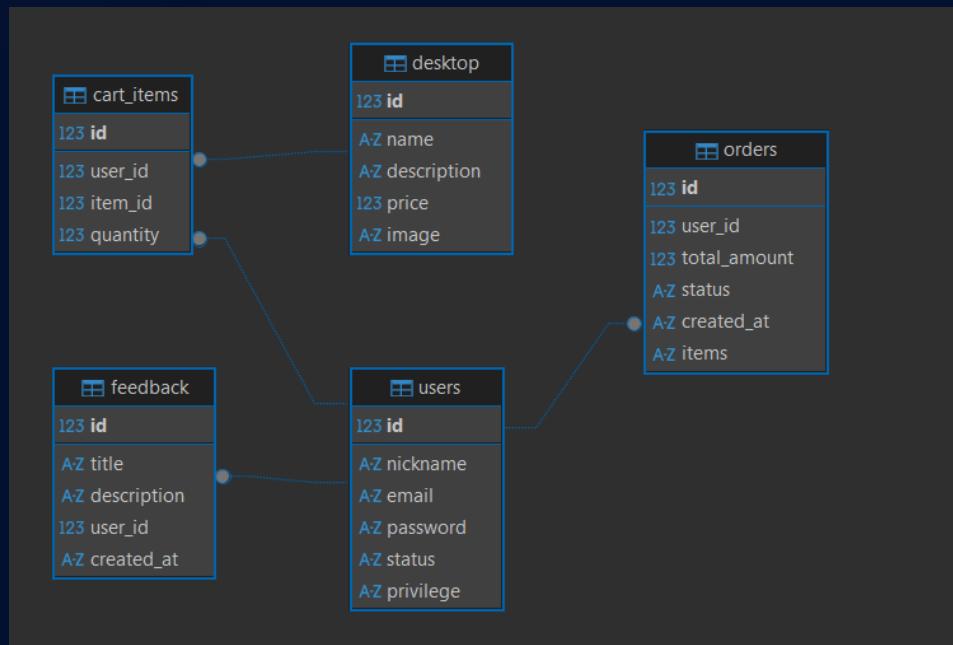
- Ізоляція: Жодних конфліктів залежностей ("It works on my machine" – у минулому).
- Легкий деплой: Розгортання однією командою `docker-compose up`.

The screenshot shows the Docker Swarm UI interface. At the top, there are two performance metrics: 'Container CPU usage' (3.77% / 1200%) and 'Container memory usage' (126.74MB / 7.22GB). Below these are search and filter controls, including a 'Search' input field and a toggle for 'Only show running containers'. The main table lists three running containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
<input type="checkbox"/>	ne_programisty	-	-	-	3.77%	13 seconds ago	
<input type="checkbox"/>	shop_container	42231d3b1036	<a href="#">ne_programisty-web</a>		3.77%	13 seconds ago	
<input type="checkbox"/>	shop_nginx	d55d358641aa	<a href="#">nginx:alpine</a>	<a href="#">80:80</a>	0%	13 seconds ago	

# СТРУКТУРА БАЗИ ДАННИХ

Використовується реляційна модель даних (SQLite).



## Основні сутності:

- 👤 **Users**: id, nickname, email, password\_hash, role.
- 💻 **Desktop (Products)**: id, name, price, image.
- 🛒 **CartItems**: Зв'язок User <-> Desktop (Many-to-One).
- 📋 **Orders**: Історія покупок, статус, сума.
- 💬 **Feedback**: Відгуки користувачів.
- 📰 **News**: Новини магазину.

Зв'язки: Користувачі мають кошики та замовлення; Відгуки прив'язані до користувачів.

# API ENDPOINTS

The screenshot displays a Swagger UI interface for a RESTful API. The left sidebar lists three main sections: Desktops, Feedbacks, and Orders. Each section contains several API endpoints with their methods, URLs, descriptions, and corresponding operation IDs.

- Desktops:**
  - GET /api/v1/desktops: Отримати список всіх настолок (Operation ID: get\_api\_v1\_desktops)
  - POST /api/v1/desktops: Додати настолку (Тільки Адмін) (Operation ID: post\_api\_v1\_desktops)
  - GET /api/v1/desktops/{desktop\_id}: Отримати деталі однієї настолки (Operation ID: get\_api\_v1\_desktops\_desktop\_id\_)
  - PATCH /api/v1/desktops/{desktop\_id}: Редагувати настолку (Тільки Адмін) (Operation ID: patch\_api\_v1\_desktops\_desktop\_id\_)
  - DELETE /api/v1/desktops/{desktop\_id}: Видалити настолку (Тільки Адмін) (Operation ID: delete\_api\_v1\_desktops\_desktop\_id\_)
- Feedbacks:**
  - GET /api/v1/feedbacks: Отримати список всіх відгуків (Operation ID: get\_api\_v1\_feedbacks)
  - POST /api/v1/feedbacks: Залишити відгук (Авторизація) (Operation ID: post\_api\_v1\_feedbacks)
  - GET /api/v1/feedbacks/{feedback\_id}: Отримати один відгук по ID (Operation ID: get\_api\_v1\_feedbacks\_feedback\_id\_)
  - PATCH /api/v1/feedbacks/{feedback\_id}: Редагувати відгук по ID відгуку (Admin only) (Operation ID: patch\_api\_v1\_feedbacks\_feedback\_id\_)
  - DELETE /api/v1/feedbacks/{feedback\_id}: Видалити відгук (Тільки Адмін) (Operation ID: delete\_api\_v1\_feedbacks\_feedback\_id\_)
  - POST /api/v1/feedbacks/user/{user\_id}: Залишити відгук (Тільки Адмін) (Operation ID: post\_api\_v1\_feedbacks\_user\_user\_id\_)
- Orders:**
  - GET /api/v1/orders/: Отримати список усіх замовлень (Тільки Адмін) (Operation ID: get\_api\_v1\_orders\_)
  - GET /api/v1/orders/my: Отримати історію своїх замовлень (Operation ID: get\_api\_v1\_orders\_my\_)
  - POST /api/v1/orders: Створити замовлення з кошика (Operation ID: post\_api\_v1\_orders\_)

The right side of the interface shows the "Scenario3 - Run results" section, which details a single test run. It includes a summary table and a detailed log of individual test cases with their status (PASS or FAIL).

Source	Environment	Iterations	Duration	All tests	Errors	Avg. Resp. Time
Runner	Flask	1	1s 672ms	9	0	50 ms

RUN SUMMARY

- POST 1.RegistAdmin: PASS Admin registered
- POST 2.LoginAdmin: PASS Admin login successful
- POST 3.CreateNews: PASS News created (201 Created)
- GET 4.GetAllNews: PASS News ID extracted
- PATCH 5.UpdateNews: PASS News updated successfully
- GET 6.ReadNews: PASS Read news success  
PASS Changes applied
- DELETE 7.DeleteNews: PASS News deleted
- GET 8.Verify404: PASS News is gone (404)

Ми розробили RESTful API, документоване через Swagger UI.

## Приклади:

- POST /api/v1/auth/login – Отримання токена.
- GET /api/v1/desktops – Отримати каталог товарів.
- POST /api/v1/carts – Додати товар у кошик.
- GET /api/v1/orders/my – Перегляд історії замовлень.
- DELETE /api/v1/users/{id} – Видалення користувача (Admin only).

# ТЕХНІЧНІ ВІЛКЛІКИ

## Проблеми та рішення:

1.  **Infinite Reload Loop:** Фронтенд зациклювався при невалідному токені.  
 **Рішення:** Виправлено ендпоїнт (/carts замість /cart) та додано Circuit Breaker в JS (sessionStorage flag).
2.  **Permissions у Docker:** Помилка запису в БД при локальному запуску скриптом app.py.  
 **Рішення:** Реалізовано "розумний" конфіг, який визначає середовище і підставляє абсолютний шлях до БД.
3.  **Nginx Static Files:** Проблема з відображенням картинок через проксі.  
 **Рішення:** Правильне налаштування Volumes та директиви proxy\_pass у nginx.conf.

# ЦІКАВІ ТЕХНІЧНІ РІШЕННЯ

## 1. Service Layer Pattern:

Винесення бізнес-логіки з роутів у сервіси. Це робить код чистим і легким для тестування.

## 2. Docker Multi-stage Build:

Використання `python:3.11-alpine` та поетапної збірки дозволило зменшити розмір образу до ~120 MB, що пришвидшує деплой.

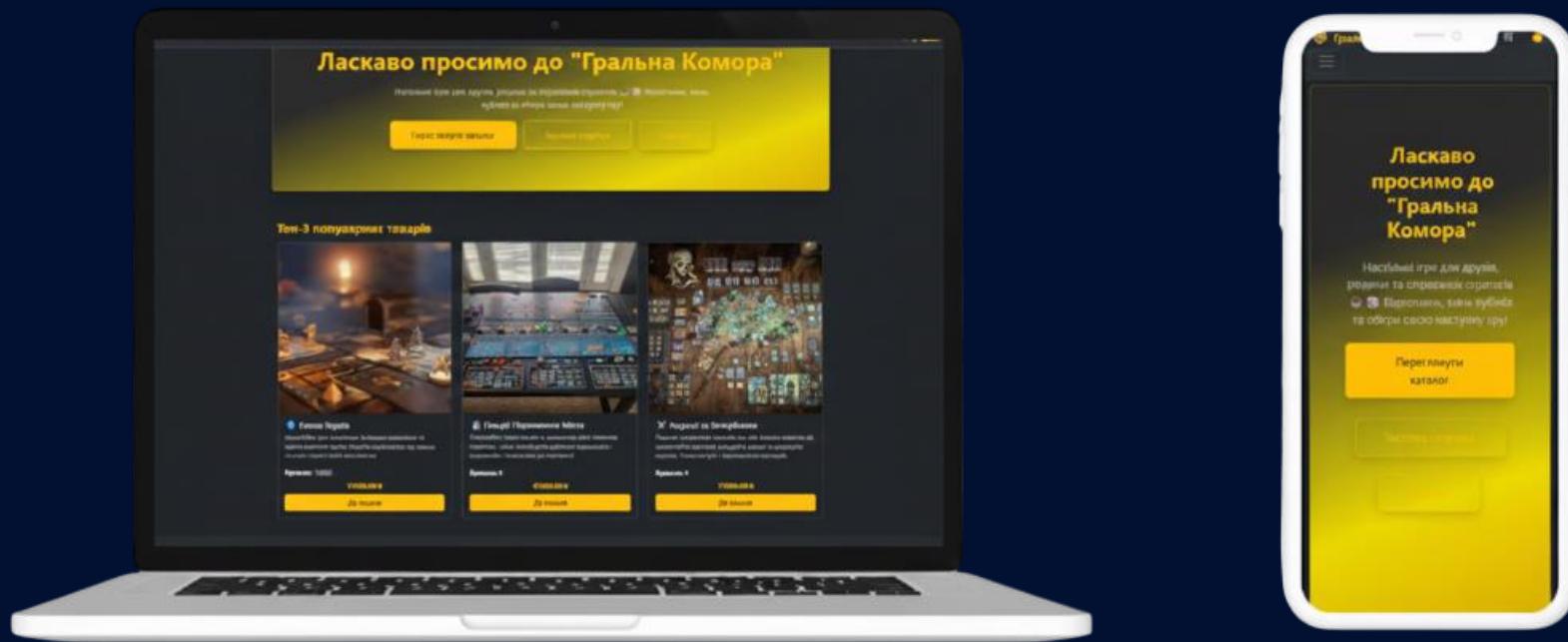
```
@api.route("/carts/clear", methods=["DELETE"])
@jwt_required()
def clear_cart_endpoint():
    # 1. Отримання ID користувача з токена
    current_user_id = get_jwt_identity()

    # 2. Виклик сервісної функції
    # Припускаємо, що CartService - це клас, який містить clear_cart
    try:
        CartService.clear_cart(user_id=current_user_id)

        # 3. Успішна відповідь
        return jsonify({
            "message": "Ваш кошик успішно очищено.",
            "user_id": current_user_id
        }), 200
    except Exception as e:
        # Обробка можливих помилок бази даних або сервісу
        print(f"Помилка при очищенні кошика користувача {current_user_id}: {e}")
        return jsonify({"error": "Не вдалося очистити кошик через внутрішню помилку сервера."}), 500
```

```
1  # === ЕТАП 1: Будівельник (Builder) ===
2  FROM python:3.11-alpine AS builder
3
4  # Робоча папка
5  WORKDIR /app
6
7  # Встановлюємо компілятори, потрібні для
8  # Це потрібно, щоб pip install не впав з
9  RUN apk add --no-cache gcc musl-dev libf
10
11 # Копіюємо список бібліотек
12 COPY requirements.txt .
13
14 # Встановлюємо залежності в user site-па
15 RUN pip install --no-cache-dir --user -r
16
17 # === ЕТАП 2: Фінальний (Final) ===
18 FROM python:3.11-alpine
19
20 # Щоб Python показував логи відразу
21 ENV PYTHONUNBUFFERED=1
22
23 # Робоча папка
24 WORKDIR /app
25
26 # Копіюємо встановлені бібліотеки з етапу
27 COPY --from=builder /root/.local /root/.
28 ENV PATH=/root/.local/bin:$PATH
29
30 # Встановлюємо wget для healthcheck
31 RUN apk add --no-cache wget
32
33 # Копіюємо ВЕСЬ код проекту в контейнер
34 COPY . .
35
36 # Підвантажуємо змінні середовища з .env
37 RUN pip install --no-cache-dir python-dotenv
```

# ДЕМОНСТРАЦІЯ



# МОЖЛИВОСТІ РОЗВИТКУ

## Roadmap:

1.  REST-Apі: Перехід всього сайту на АПІ-запити
2.  PostgreSQL: Міграція з SQLite для кращої продуктивності під навантаженням.
3.  Redis: Кешування запитів API та сесій.
4.  Платіжна система: Інтеграція LiqPay або Stripe.
5.  HTTPS: Налаштування Certbot (Let's Encrypt) у Nginx-контейнері.
6.  Mobile App: Розробка повноцінного мобільного застосунку, використовуючи наш готовий API.

# ВІДНОВКИ

## Результати:

Ми пройшли повний шлях SDLC (Software Development Life Cycle): від ідеї та макетів до контейнеризованого продукту, готового до деплою в хмару (Render.com).

## Здобуті навички:

- Робота з **Git Flow** та вирішення конфліктів.
- Проєктування **REST API** та баз даних.
- Написання **Unit & Integration** тестів.
- **DevOps**: Docker, Nginx, CI/CD основи.



**Проект "Гральна Комора" готовий до масштабування!**  
**Дякуємо за увагу!**