SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Ústav informatiky a matematiky

Dokumentácia k semestrálnemu zadaniu Neighbourhood portal

Semestrálne zadanie

Bc. Kňazovický Marián, Bc. Hrobárová Ema, Bc. Juraj Revaj,
Bc. Belko Andrej, Bc. Adamcová Kristína

Bratislava 2024

Obsah

Úvod

O aplikácii

Neighborhood Portal je progresívna webová aplikácia (PWA) navrhnutá na uľahčenie interakcie medzi komunitami. Umožňuje susedom zdieľať služby a predávať produkty v rámci miestnej komunity.

Použité technológie

V rámci implementácie projektu boli použité nasledujúce hlavné technológie:

1. **Next.js –** framework postavený na React.js, ktorý umožňuje vytvárať webové aplikácie s server-side renderingom a statickou generáciou stránok. Poskytuje optimalizovanú architektúru pre výkon a SEO.
2. **React –** knižnica vyvinutá spoločnosťou Meta (predtým Facebook) pre tvorbu používateľských rozhraní. Umožňuje vytváranie komponentovo orientovaných aplikácií s efektívnym DOM renderovaním.
3. **NextAuth.js –** komplexné riešenie pre autentifikáciu v Next.js aplikáciách, ktoré poskytuje flexibilné možnosti zabezpečenia a správy používateľských relácií.
4. **PostgreSQL -** Výkonný open-source relačný databázový systém s dôrazom na rozšíriteľnosť a SQL kompatibilitu. Poskytuje robustné riešenie pre ukladanie a správu dát s podporou pokročilých funkcií a transakcií.
5. **Prisma –** moderný ORM (Object-Relational Mapping) nástroj pre Node.js a TypeScript, ktorý zjednodušuje prácu s databázou a poskytuje typovú bezpečnosť.
6. **Docker –** platforma pre vývoj, prepravu a spúšťanie aplikácií pomocou kontajnerizácie. Umožňuje zabaliť aplikáciu so všetkými jej závislosťami do štandardizovaného kontajnera, čím zabezpečuje konzistentné prostredie naprieč rôznymi systémami.
7. **Git –** verziovací systém

Ďalšie použité knižnice:

- next-pwa
- bcrypt
- zod
- jest

Architektúra

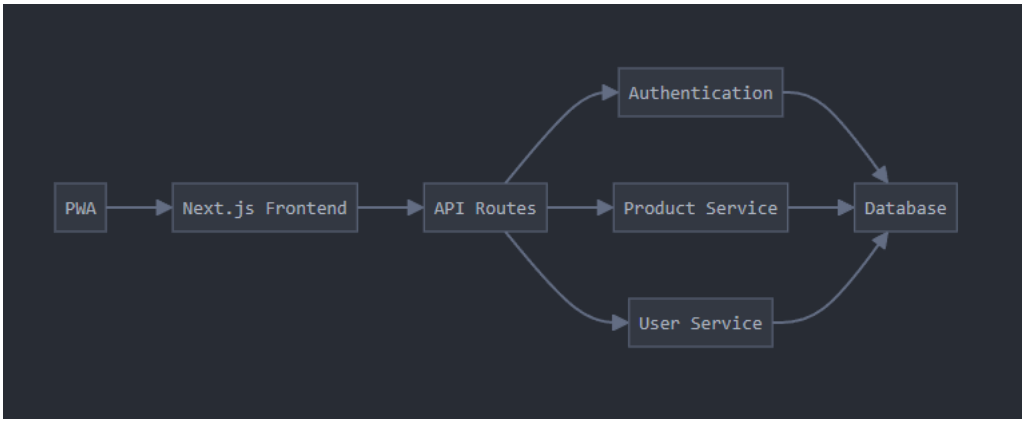
Aplikácia naplno využíva možnosti Fullstack frameworku Next.js. Použili sme verziu s „app router“ priečinkom. Next.js ma presne definovanú adresárovú a súborovú štruktúru vďaka ktorej rovno vytvára podstránky alebo API endpointy. Na ukladanie všetkých dôležitých systémových premenných sme použili .env súbor, ktorý si treba s vlastnými hodnotami vytvoriť pre správne lokálne spustenie aplikácie. Všetky formuláre v systéme sú validované na strane klienta aj servera.

Frontend

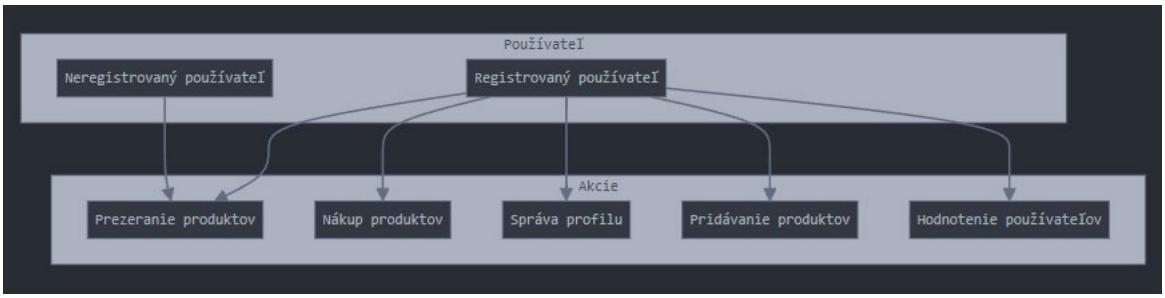
- **Framework**: Next.js 14
- **Štýly**: TailwindCSS

Backend

- **API**: Next.js API Routes
- **Databázový ORM**: Prisma
- **Databáza**: PostgreSQL
- **Autentifikácia**: NextAuth.js s JWT tokenmi



Obrázok 1 Component diagram zobrazujúci vzťahy medzi komponentmi aplikácie



Obrázok 2 Use Case diagram zobrazujúci možnosti rôznych typov používateľov

Docker

Aplikácia využíva Docker pre kontajnerizáciu s dvomi hlavnými službami:

- Databázový servis (PostgreSQL)
- Aplikačný servis (Next.js)

Konfigurácia je definovaná v súbore docker-compose.yaml.

```yaml
services:
  db:
    image: postgres:latest
    ports:
      - '${POSTGRES_PORT}:5432'
    environment:
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_DB=${POSTGRES_DB}
    healthcheck:
      test: ["CMD-SHELL", "sh -c 'pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}'"]
      interval: 10s
      timeout: 3s
      retries: 3
    networks:
      - asos-network

  app:
    build: .
    ports:
      - '3000:3000'
    volumes:
      - .:/app
      - /app/node_modules
      - /app/.next
    depends_on:
      db:
        condition: service_healthy
    environment:
      - DATABASE_URL=postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@db:${POSTGRES_PORT}/${POSTGRES_DB}
    networks:
      - asos-network
    command: ["sh", "-c", "npx prisma migrate dev --name init --create-only && npx prisma migrate deploy && npx prisma db seed && npm run dev"]

networks:
  asos-network:
    driver: bridge
```

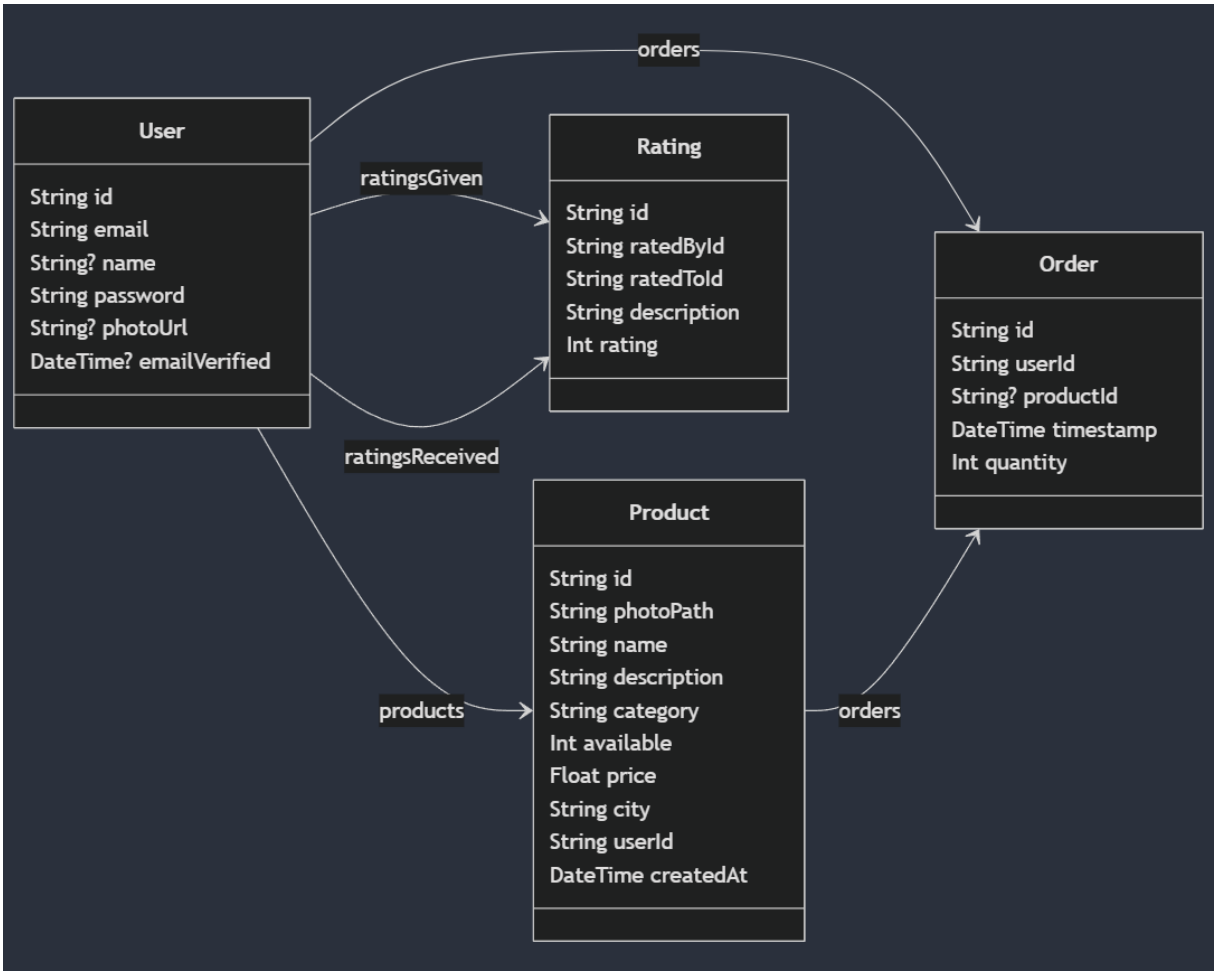
*Obrázok 3 Docker konfigurácia definovaná v **docker-compose.yaml***

Funkcionality systému

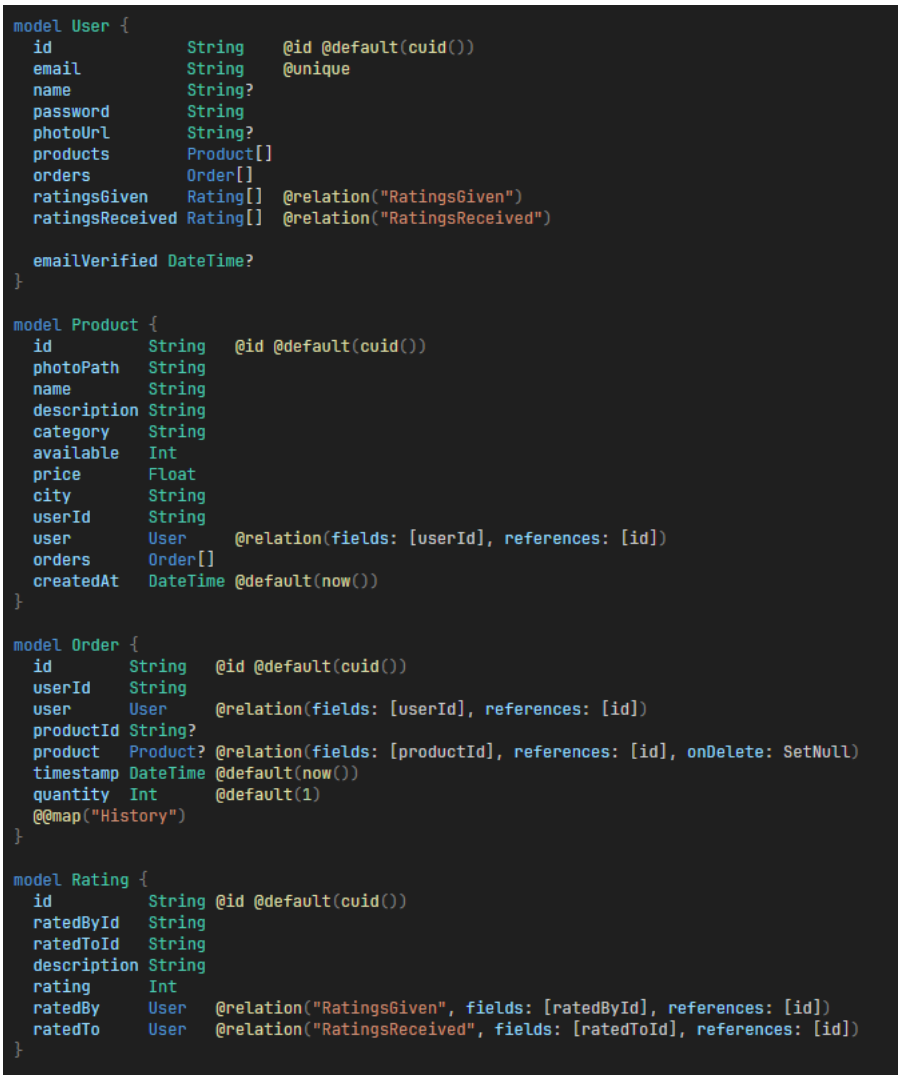
Aplikácia poskytuje komplexné riešenie pre správu online obchodu s rôznymi funkcionalitami pokrývajúcimi všetky aspekty elektronického obchodovania. Nižšie sú popísané hlavné funkcionality systému a ich kľúčové komponenty.

- **Autentifikácia používateľov** Komplexný systém správy používateľských účtov zabezpečujúci bezpečný prístup k aplikácii. Implementuje moderné bezpečnostné praktiky vrátane šifrovania citlivých údajov.
    - Šifrovanie hesiel pomocou bcrypt
    - Overenie sily hesla
- **Správa produktov** Rozsiahly systém pre správu produktového katalógu umožňujúci predajcom efektívne spravovať svoj tovar. Poskytuje intuitívne rozhranie pre všetky potrebné operácie s produktami.
    - CRUD operácie
    - Podpora pre nahrávanie obrázkov
    - Filtrovanie podľa kategórií
- **Používateľské profily** Personalizované používateľské rozhranie umožňujúce správu osobných údajov a sledovanie aktivít v systéme. Poskytuje komplexný prehľad o všetkých interakciách používateľa s platformou.
    - Správa osobných informácií
    - Systém hodnotenia
    - História transakcií
    - História ponúkaných produktov
- **Systém objednávok** Zabezpečuje plynulý proces nákupu.

Databázová schéma



Obrázok 4 UML class diagram popisujúci databázovú schému



```
model User {
  id              String    @id @default(cuid())
  email           String    @unique
  name            String?
  password        String
  photoUrl        String?
  products        Product[]
  orders          Order[]
  ratingsGiven    Rating[]  @relation("RatingsGiven")
  ratingsReceived Rating[]  @relation("RatingsReceived")

  emailVerified DateTime?
}

model Product {
  id          String   @id @default(cuid())
  photoPath   String
  name        String
  description String
  category    String
  available   Int
  price       Float
  city        String
  userId      String
  user        User     @relation(fields: [userId], references: [id])
  orders      Order[]
  createdAt   DateTime @default(now())
}

model Order {
  id        String   @id @default(cuid())
  userId    String
  user      User     @relation(fields: [userId], references: [id])
  productId String?
  product   Product? @relation(fields: [productId], references: [id], onDelete: SetNull)
  timestamp DateTime @default(now())
  quantity  Int      @default(1)
  @@map("History")
}

model Rating {
  id          String @id @default(cuid())
  ratedById   String
  ratedToId   String
  description String
  rating      Int
  ratedBy     User   @relation("RatingsGiven", fields: [ratedById], references: [id])
  ratedTo     User   @relation("RatingsReceived", fields: [ratedToId], references: [id])
}
```

*Obrázok 5 Databázová schéma zapísaná v súbore **schema.prisma***

API Dokumentácia

Aplikácia poskytuje REST API rozhranie, ktoré umožňuje interakciu s rôznymi časťami systému. API endpointy sú rozdelené do logických skupín podľa ich funkcionality. Jednotlivé endpointy síce sú implementované avšak Next.js v kombinácii s Prisma ORM nevyžaduje vytváranie jednotlivých endpointov. Best practice je vytváranie asynchrónnych funkcií, ktoré pomocou prisma client získajú dáta z databázy, ktoré následne vrátia ako návratovú hodnotu. Kvôli splneniu zadaných požiadaviek sme však aj implementovali jednotlivé endpointy. Všetky vracajú dáta v JSON formáte.

Autentifikačné endpointy

Slúžia na správu používateľských účtov a autentifikáciu v systéme. O autentifikáciu sa stará knižnica **next-auth**, ktorá sa stará o prihlasovanie, odhlasovanie a správu tokenu používateľa.
V konfiguračných súboroch sa akurát definujú tzv. *callback* funkcie, v ktorých sa nachádzajú naše požadované akcie (napr. spracovanie prihlasovania atď...).

- **POST /api/auth/register**
    - Registrácia nového používateľa
    - Vyžaduje email, meno používateľa a 2x zadané rovnaké silné heslo
    - Po úspešnej registrácii presmeruje na prihlásenie
- **POST /api/auth/login**
    - Prihlásenie používateľa do systému
    - Vyžaduje email a heslo
    - Vracia JWT token pre ďalšiu autentifikáciu
- **POST /api/auth/logout**
    - Odhlásenie používateľa zo systému
    - Invaliduje aktuálny JWT token

Produktové endpointy

Umožňujú správu produktov v systéme.

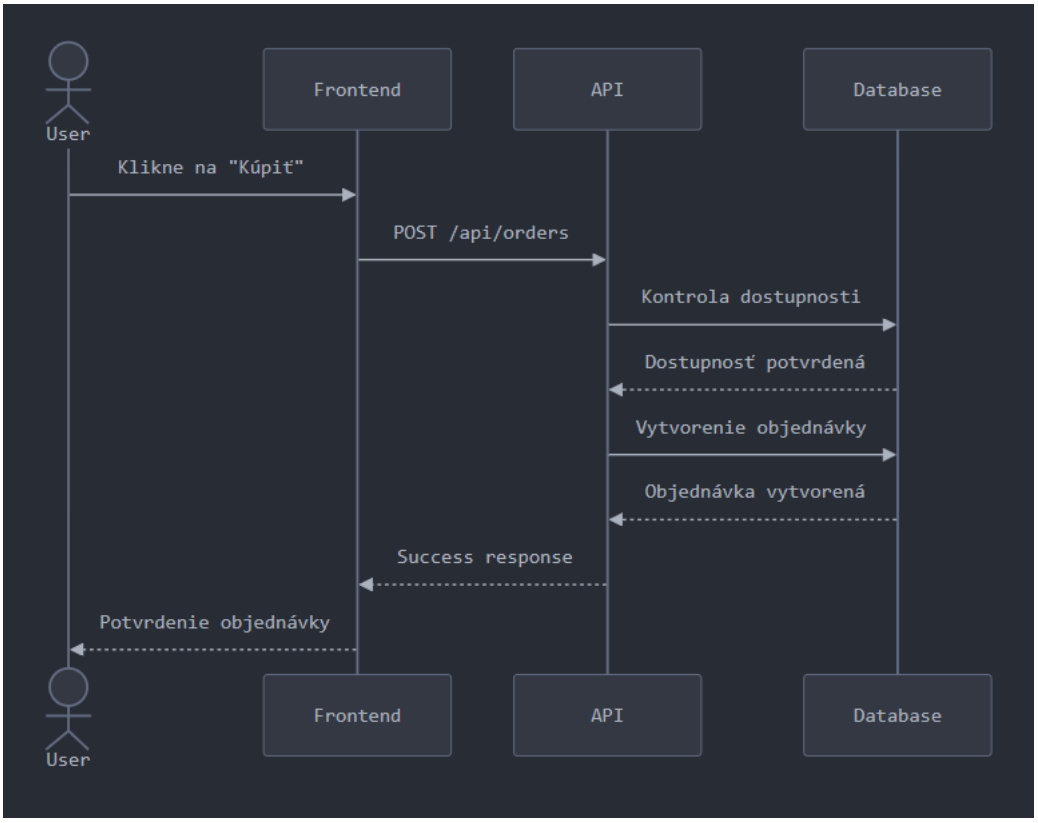
- **GET /api/products**
    - Získanie zoznamu všetkých produktov
    - Podporuje filtrovanie podľa kategórie a mesta
    - Podporuje stránkovanie a zoradenie
- **GET /api/products/:id**
    - Získanie detailných informácií o konkrétnom produkte
    - Vracia všetky údaje vrátane informácií o predajcovi
- **POST /api/products**
    - Vytvorenie nového produktu
    - Vyžaduje autentifikáciu používateľa
    - Podporuje nahrávanie obrázkov
- **PUT /api/products/:id**
    - Aktualizácia existujúceho produktu
    - Dostupné len pre vlastníka produktu
    - Možnosť aktualizácie všetkých atribútov
- **DELETE /api/products/:id**
    - Odstránenie produktu zo systému
    - Dostupné len pre vlastníka produktu

Používateľské endpointy

Poskytujú prístup k používateľským profilom a hodnoteniam.

- **GET /api/users/:id**
    - Získanie profilu používateľa
    - Vracia verejné informácie o používateľovi
    - Zahŕňa štatistiky predaja a nákupov
- **PUT /api/users/:id**
    - Aktualizácia profilu používateľa
    - Dostupné len pre vlastníka profilu
    - Možnosť zmeny osobných údajov a fotografie
- **GET /api/users/:id/ratings**
    - Získanie hodnotení používateľa
    - Obsahuje prijaté aj odoslané hodnotenia

- Podporuje filtrovanie a stránkovanie



Obrázok 6 Sekvenčný diagram zobrazujúci postupnosť akcií pri objednávke produktu

Bezpečnosť

Aplikácia implementuje viacero bezpečnostných mechanizmov na zabezpečenie dát a ochranu používateľov. Nižšie sú popísané hlavné bezpečnostné prvky systému.

Autentifikácia

Systém využíva moderné bezpečnostné praktiky pre správu používateľských účtov a prístupu:

- **JWT (JSON Web Token) autentifikácia**
  - Bezstavová autentifikácia pomocou tokenov
  - Časovo obmedzená platnosť tokenov
  - Bezpečné ukladanie tokenov v HTTP-only cookies
- **Hashovanie hesiel**
  - Využitie bcrypt algoritmu pre hashovanie hesiel
  - Implementácia salt rounds pre zvýšenú bezpečnosť
  - Bezpečné ukladanie hashov v databáze
- **Validácia vstupov pomocou Zod schém**

```
export const loginSchema = object({
    email: string({ required_error: "Email is required" })
        .min(1, "Email is required")
        .email("Invalid email"),
    password: string({ required_error: "Password is required" })
        .min(1, "Password is required"),
});

export const registerSchema = object({
    email: z.string().email("Invalid email address"),
    name: z.string().min(1, "Name is required"),
    password: z
        .string()
        .min(8, "Password must be at least 8 characters long")
        .regex(/[A-Z]/, "Password must contain at least one uppercase letter")
        .regex(/[a-z]/, "Password must contain at least one lowercase letter")
        .regex(/\d/, "Password must contain at least one number")
        .regex(/[@$!%*?&]/, "Password must contain at least one special character"),
});
```

Obrázok 7 Validácia prihlasovacích a registračných údajov

Ochrana dát

Implementované bezpečnostné opatrenia pre ochranu pred bežnými typmi útokov:

- **CSRF (Cross-Site Request Forgery) ochrana**
  - Generovanie jedinečných CSRF tokenov
  - Validácia tokenov pri každej požiadavke
  - Implementácia SameSite cookies
- **Prevencia XSS (Cross-Site Scripting)**
  - Automatické escapovanie nebezpečného obsahu
  - Content Security Policy (CSP) hlavičky
  - Validácia všetkých používateľských vstupov

Testovanie

Aplikácia využíva komplexné testovanie komponentov a funkcionalít pomocou moderných testovacích nástrojov a metodík.

Použité technológie

- Jest - testovací framework
- React Testing Library - knižnica pre testovanie React komponentov
- Mock objekty - pre simuláciu externých závislostí

Typy testov

- **Unit testy**
  - Testovanie jednotlivých komponentov
  - Overenie správneho renderovania
  - Kontrola správania pri rôznych vstupných dátach
- **Integračné testy**
  - Testovanie interakcií medzi komponentmi
  - Overenie správneho toku dát
  - Simulácia používateľských akcií