CinemaApp նախագիծ

Այս փաստաթուղթը մանրամասն ներկայացնում է իմ CinemaApp անհատական նախագիծը՝ սկսած տվյալների բազայի ստեղծման գործընթացից մինչև JavaScript-ում հավելվածի կառուցումն ու գործարկումը: Ամեն փուլում ես բացատրում եմ, թե ինչպես և ինչն ինչպիսին իրագործվեց։

# 1. Տվյալների բազայի ստեղծում

Առաջին քայլում ես տեղադրում և ինտեգրել եմ MySQL Server: Օգտագործեցի Ubuntu-ում `sudo apt-get install mysql-server` հրամանը, ապա կատարեցի հիմնական կարգավորումները՝ root օգտահաշիվը պաշտպանելու համար։ Ստեղծեցի `cinemaapp` տվյալների բազան՝ հետեւյալ հրահանգով՝

```sql  
CREATE DATABASE cinemaapp;  
USE cinemaapp;  
```

Գործընթացի ընթացքում ես ամբողջությամբ նշել եմ թարմացումները՝ ապահովելով UTF-8 կոդավորումը և անհրաժեշտ պարամետրերը։

# 2. Աղյուսակների կառուցում և տվյալների ներմուծում

Հաջորդը կազմեցի SQL սքրիպտերը, որի միջոցով ստեղծեցի երեք հիմնական աղյուսակ՝ `movies`, `persons` եւ `bookings`. Սքրիպտի օրինակ՝

```sql  
CREATE TABLE movies (  
 id INT AUTO\_INCREMENT PRIMARY KEY,  
 title VARCHAR(255) NOT NULL,  
 poster VARCHAR(255),  
 genres VARCHAR(255),  
 release\_date DATE,  
 average\_score FLOAT,  
 overview TEXT  
);  
...  
```

Այդուհետ ես խմբագրեցի և ընտեցի նմուշային տվյալները՝ ամենակարեւոր ֆիլմերի վերաբերյալ, օրինակ՝ „Inception“-ը, և օբյեկտիվ մանրամասներ ներմեցի հետևյալ հարցմամբ՝

```sql  
INSERT INTO movies (...) VALUES ('Inception', ...);  
INSERT INTO persons (...) VALUES ('Meri', 'Davtyan', ...);  
INSERT INTO bookings (...) VALUES (1, 'Yerevan', 'hy', '2025-05-20 18:30:00', '["A1","A2"]', 1);  
```

# 3. Բեքէնդի զարգացում JavaScript-ով

Ես ընտեցի Node.js և Express.js՝ REST API endpoints մշակելու համար: Ծրագրում օգտագործեցի `mysql2` գրադարանը, որը հնարավորություն է տալիս promise-եր async/await սինթաքսով օգտագործելու: Ես գրել եմ `db/index.js` ֆայլը, ուր տեղադրել եմ հետևյալ առկայությունները.

```js  
const mysql = require('mysql2/promise');  
// pool կոնֆիգի կոդ  
async function getBookings(req, res) {  
 const [rows] = await pool.query(...);  
 res.json(rows.map(r => JSON.parse(r.seats)));   
}  
```

Այս կոդի միջոցով ես հեշտությամբ ընտրում և վերակառուցում եմ նստատեղերի տվյալները՝ JSON տողից JavaScript օբյեկտ: Տարանցման սխալները ստուգեցի try/catch բլոկներում, և օգտագործեցի կոմիտ/ռոլբեք հոսքեր:

# 4. Ֆրոնթէնդի կառուցում React.js-ով

Ֆրոնթէնդի համար ես ընտրեցի React.js: Գործարկեցի CRA (Create React App) կամ կարգավորեցի Webpack+Babel: Կազմեցի մինիմալ UI՝ React Components-ներով՝ օգտագործելով `react-numeric-input`՝ նստատեղերի քանակի ընտրության համար, `redux-form` կամ hooks`-ների միջոցով ֆորմերի կառավարման համար։

Ես կառուցել եմ հաջորդ կոմպոնենտները՝ `MovieList`, `MovieDetail`, `BookingForm` և `Confirmation`: MovieList–ում ես լցնում եմ ֆիլմերի տվյալները backend-ից, MovieDetail–ում ցուցադրում եմ տվյալ ֆիլմի մանրամասները, BookingForm–ում օգտվողը ընտրում է նստատեղերը, և վերջում Confirmation կցուցադրում է ամրացված տվյալները։

# 5. Թեստավորում և ներդրում

Ծրագրի ընթացքում ես օգտագործել եմ Postman API endpoints ստուգելու համար. Ստուգել եմ GET և POST հարցումները, ըստ response.status կոդերի և JSON ֆորմատի: Ֆրոնթէնդի աշխատանքի տիկշները տեսա Chrome DevTools-ի միջոցով, և կատարելագործեցի սխալ կառավարման մոտեցումները՝

```js  
app.use((err, req, res, next) => {  
 console.error(err);  
 res.status(500).json({ error: 'Something went wrong' });  
});  
```

Վերջնական շրջանում ես սերվեր եմ տեղադրել Heroku կամ AWS վրա, առնվազն տեղական Docker կոնտեյներների միջոցով: Եվ ստուգեցի, որ կայքը մատչելի է բազմաթիվ սարքերի վրա։

# 6. Եզրակացություն

Այս նախագիծը հնարավորություն տվեց ինքս գնահատել ամբողջական վեբ հավելված ստեղծելու փուլային մոտեցումը՝ տվյալների բազայի, բեքէնդի, ֆրոնթէնդի և տեղաբաշխման չափերով: Ես ձեռք բերեցի արժեքավոր փորձ Node.js, MySQL, React.js և DevOps միջոցառումների մեջ։