Full-Stack Web Development

O'zbek tilida o'rganish kitobi

Boshlanuvchidan professional darajagacha

2024 yil

Mundarija

1. Kirish va Web Development Asoslari

Web Development nima?

Web Development - bu veb-saytlar va veb-ilovalarni yaratish san'atidir. Bu soha ikki asosiy qismga bo'linadi:

Frontend Development: Foydalanuvchi ko'radigan va ular bilan o'zaro ta'sirda bo'ladigan qismni yaratish. Bu HTML, CSS va JavaScript texnologiyalarini o'z ichiga oladi.

Backend Development: Server tomonini, ma'lumotlar bazasini va biznes mantiqini yaratish. Bu Node.js, Python, PHP kabi texnologiyalardan foydalanadi.

Full-Stack Development: Ham frontend, ham backend tomonlarini bilish va ishlay olish. Bu kitobda siz full-stack developer bo'lishni o'rganasiz.

Web development sohasida muvaffaqiyat qozonish uchun quyidagi ko'nikmalar kerak:
- HTML/CSS asoslari
- JavaScript dasturlash tili
- Frontend framework'lar (React, Vue, Angular)
- Backend texnologiyalar (Node.js, Express)
- Ma'lumotlar bazasi (MongoDB, PostgreSQL)
- Version control (Git)
- Deployment va hosting

```
<!DOCTYPE html>
<html lang="uz">
<head>
    <meta charset="UTF-8">
    <title>Mening birinchi sahifam</title>
</head>
<body>
    <h1>Salom Dunyo!</h1>
    <p>Bu mening birinchi veb-sahifam.</p>
</body>
</html>
```

Amaliy mashq:

Yuqoridagi HTML kodini o'z kompyuteringizda yarating va brauzerda oching.

Development Environment sozlash

Professional web development uchun quyidagi vositalar kerak:

1. Code Editor:
   - Visual Studio Code (tavsiya etiladi)
   - Sublime Text
   - Atom

2. Web Browser:
   - Google Chrome (Developer Tools bilan)
   - Firefox Developer Edition

  - Safari (Mac uchun)

3. Node.js:
   - JavaScript runtime environment
   - npm (Node Package Manager) bilan birga keladi
   - Serverda JavaScript ishlatish imkonini beradi

4. Git:
   - Version control system
   - Kodingizni saqlash va boshqarish uchun
   - GitHub, GitLab bilan integratsiya

5. Terminal/Command Line:
   - Windows: Command Prompt yoki PowerShell
   - Mac/Linux: Terminal
   - Git Bash (Windows uchun)

Qo'shimcha vositalar:
- Postman (API testing uchun)
- MongoDB Compass (database management)
- Chrome DevTools
- Extensions: Live Server, Prettier, ESLint

Amaliy mashq:

Visual Studio Code'ni o'rnating va birinchi HTML faylini yarating.

2. HTML - Veb-sahifalarning Asosi

HTML asoslari va strukturasi

HTML (HyperText Markup Language) - veb-sahifalarning asosiy strukturasini yaratuvchi markup tilidir.

HTML elementlari:
- Tag'lar: <tagname>content</tagname>
- Attributes: <tag attribute="value">
- Nested elements: bir element ichida boshqa element

Asosiy HTML strukturasi:
- DOCTYPE declaration
- html elementi
- head elementi (meta ma'lumotlar)
- body elementi (ko'rinadigan kontent)

Muhim HTML elementlari:
- Headings: h1, h2, h3, h4, h5, h6
- Paragraphs: p
- Links: a
- Images: img
- Lists: ul, ol, li
- Divisions: div
- Spans: span
- Forms: form, input, textarea, button

Semantic HTML:
- header, nav, main, section, article, aside, footer
- Bu elementlar SEO va accessibility uchun muhim

```html
<!DOCTYPE html>
<html lang="uz">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML Asoslari</title>
</head>
<body>
    <header>
        <h1>Mening Veb-saytim</h1>
        <nav>
            <ul>
                <li><a href="#home">Bosh sahifa</a></li>
                <li><a href="#about">Haqida</a></li>
                <li><a href="#contact">Aloqa</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <section id="home">
            <h2>Xush kelibsiz!</h2>
            <p>Bu mening birinchi professional veb-saytim.</p>
            <img src="welcome.jpg" alt="Xush kelibsiz rasmi">
        </section>
        <section id="about">
            <h2>Men haqimda</h2>
            <p>Men web developer bo'lishni o'rganyapman.</p>
        </section>
    </main>

    <footer>
        <p>&copy; 2024 Mening Veb-saytim. Barcha huquqlar himoyalangan.</p>
    </footer>
</body>
</html>
```

Amaliy mashq:

Yuqoridagi kodni o'zgartirib, o'zingiz haqingizda sahifa yarating.

HTML Forms va Input elementlari

HTML formalar foydalanuvchilardan ma'lumot olish uchun ishlatiladi.

Form elementlari:
- form: forma konteyner
- input: turli xil ma'lumot kiritish
- textarea: ko'p qatorli matn
- select: dropdown ro'yxat
- button: tugma
- label: input uchun yorliq

Input turlari:
- text: oddiy matn
- email: email manzil
- password: parol
- number: raqam
- date: sana
- checkbox: belgilash katakchasi
- radio: tanlov tugmasi
- file: fayl yuklash
- submit: forma yuborish

Form attributes:
- action: forma qayerga yuboriladi
- method: GET yoki POST
- required: majburiy maydon
- placeholder: ko'rsatma matni
- value: standart qiymat

```
<form action="/submit" method="POST">
    <div>
        <label for="name">Ismingiz:</label>
        <input type="text" id="name" name="name" required>
    </div>

    <div>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
    </div>

    <div>
        <label for="age">Yoshingiz:</label>
        <input type="number" id="age" name="age" min="1" max="120">
    </div>

    <div>
        <label for="message">Xabar:</label>
        <textarea id="message" name="message" rows="4" cols="50"></textarea>
    </div>

    <div>
        <label>
            <input type="checkbox" name="subscribe" value="yes">
            Yangiliklar uchun obuna bo'lish
        </label>
    </div>

    <div>
        <label>Jinsingiz:</label>
        <input type="radio" id="male" name="gender" value="male">
        <label for="male">Erkak</label>
        <input type="radio" id="female" name="gender" value="female">
        <label for="female">Ayol</label>
    </div>

    <button type="submit">Yuborish</button>
</form>
```

Amaliy mashq:

Ro'yxatdan o'tish formasini yarating: ism, familiya, email, parol va parolni tasdiqlash maydonlari bilan.

3. CSS - Dizayn va Styling

CSS asoslari va selektorlar

CSS (Cascading Style Sheets) - HTML elementlariga stil berish uchun ishlatiladi.

CSS qo'shish usullari:
1. Inline CSS: style attribute
2. Internal CSS: <style> tag ichida
3. External CSS: alohida .css fayl

CSS selektorlar:
- Element selector: p, h1, div
- Class selector: .classname
- ID selector: #idname
- Attribute selector: [attribute="value"]
- Pseudo-class: :hover, :focus, :nth-child
- Pseudo-element: ::before, ::after

CSS properties:
- Color va background
- Font va text
- Margin va padding
- Border va outline
- Width va height
- Position va display
- Flexbox va Grid

CSS Box Model:
- Content: asosiy kontent
- Padding: kontent atrofidagi bo'sh joy
- Border: chegara
- Margin: element atrofidagi bo'sh joy

```css
/* External CSS fayl - styles.css */

/* Element selector */
body {
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f5f5f5;
}

/* Class selector */
.container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 20px;
}

.card {
    background: white;
    border-radius: 10px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
    padding: 20px;
    margin-bottom: 20px;
}

/* ID selector */
#header {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    text-align: center;
    padding: 50px 0;
}

/* Pseudo-class */
.button {
    background: #007bff;
    color: white;
    padding: 12px 24px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background 0.3s ease;
}

.button:hover {
    background: #0056b3;
}

/* Responsive design */
@media (max-width: 768px) {
    .container {
        padding: 10px;
    }

    .card {
        margin-bottom: 10px;
    }
}
```

Amaliy mashq:

Yuqoridagi CSS kodini ishlatib, chiroyli card dizaynli sahifa yarating.

Flexbox va Grid Layout

Modern CSS layout sistemlari - Flexbox va Grid.

FLEXBOX:
Bir o'lchamli layout uchun (qator yoki ustun)

Flex Container properties:
- display: flex
- flex-direction: row, column, row-reverse, column-reverse
- justify-content: flex-start, center, flex-end, space-between, space-around
- align-items: flex-start, center, flex-end, stretch
- flex-wrap: nowrap, wrap, wrap-reverse

Flex Item properties:
- flex-grow: elementning o'sish koeffitsienti
- flex-shrink: elementning qisqarish koeffitsienti

- flex-basis: elementning asosiy o'lchami
- align-self: individual alignment

CSS GRID:
Ikki o'lchamli layout uchun (qator va ustun)

Grid Container properties:
- display: grid
- grid-template-columns: ustunlar o'lchami
- grid-template-rows: qatorlar o'lchami
- grid-gap: elementlar orasidagi bo'shliq
- justify-items: gorizontal alignment
- align-items: vertikal alignment

Grid Item properties:
- grid-column: ustun pozitsiyasi
- grid-row: qator pozitsiyasi
- grid-area: maydon nomi

```
/* FLEXBOX misoli */
.flex-container {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 20px;
    background: #f8f9fa;
}

.flex-item {
    flex: 1;
    margin: 0 10px;
    padding: 20px;
    background: white;
    border-radius: 8px;
    text-align: center;
}

/* Responsive navigation */
.navbar {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 1rem 2rem;
    background: #333;
    color: white;
}

.nav-links {
    display: flex;
    list-style: none;
    margin: 0;
    padding: 0;
}

.nav-links li {
    margin-left: 2rem;
}

/* CSS GRID misoli */
.grid-container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    grid-gap: 20px;
    padding: 20px;
}

.grid-item {
    background: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

/* Complex grid layout */
.page-layout {
    display: grid;
    grid-template-areas:
        "header header header"
        "sidebar main main"
        "footer footer footer";
    grid-template-rows: auto 1fr auto;
    grid-template-columns: 250px 1fr 1fr;
    min-height: 100vh;
    grid-gap: 20px;
}

.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
.footer { grid-area: footer; }
```

Amaliy mashq:

Flexbox va Grid'dan foydalanib, responsive card layout yarating.

4. JavaScript - Dasturlash Asoslari

JavaScript asoslari va sintaksis

JavaScript - veb-sahifalarga interaktivlik qo'shish uchun ishlatiladi.

O'zgaruvchilar:
- let: block scope, qayta tayinlanishi mumkin
- const: block scope, o'zgarmas qiymat
- var: function scope (eski usul)

Ma'lumot turlari:
- Primitive: string, number, boolean, undefined, null, symbol
- Non-primitive: object, array, function

Operatorlar:
- Arithmetic: +, -, *, /, %, **
- Assignment: =, +=, -=, *=, /=
- Comparison: ==, ===, !=, !==, <, >, <=, >=
- Logical: &&, ||, !
- Ternary: condition ? true : false

Shartli operatorlar:
- if...else
- switch...case
- Ternary operator

Sikllar:
- for loop
- while loop
- do...while loop
- for...in loop
- for...of loop

```
// O'zgaruvchilar va ma'lumot turlari
let ism = "Ahmad";
const yosh = 25;
let tugilganYil = 2024 - yosh;

// Ma'lumot turlari
let son = 42;              // Number
let matn = "Salom";        // String
let togri = true;          // Boolean
let massiv = [1, 2, 3];    // Array
let obyekt = {             // Object
    ism: "Ali",
    yosh: 20,
    shahar: "Toshkent"
};

// Funksiyalar
function salomlashish(ism) {
    return "Salom, " + ism + "!";
}

// Arrow function
```

```
const yigindi = (a, b) => a + b;

// Shartli operatorlar
if (yosh >= 18) {
    console.log("Kattalar");
} else {
    console.log("Bolalar");
}

// Switch statement
switch (kun) {
    case "dushanba":
        console.log("Hafta boshi");
        break;
    case "juma":
        console.log("Juma muborak");
        break;
    default:
        console.log("Oddiy kun");
}

// Sikllar
for (let i = 0; i < 5; i++) {
    console.log("Raqam: " + i);
}

// Array bilan ishlash
let mevalar = ["olma", "banan", "apelsin"];
for (let meva of mevalar) {
    console.log(meva);
}

// Object bilan ishlash
for (let kalit in obyekt) {
    console.log(kalit + ": " + obyekt[kalit]);
}
```

Amaliy mashq:

Foydalanuvchi yoshini so'rab, 18 yoshdan katta yoki kichikligini aniqlaydigan dastur yozing.

DOM manipulation va Events

DOM (Document Object Model) - HTML elementlarini JavaScript orqali boshqarish.

Element tanlash:
- getElementById(): ID bo'yicha
- getElementsByClassName(): class bo'yicha
- getElementsByTagName(): tag bo'yicha
- querySelector(): CSS selector bo'yicha
- querySelectorAll(): barcha mos elementlar

Element o'zgartirish:
- innerHTML: ichki HTML
- textContent: matn kontenti
- style: CSS stillari
- classList: class'larni boshqarish
- setAttribute(): attribute qo'shish

- getAttribute(): attribute olish

Event'lar:
- click: bosganda
- submit: forma yuborilganda
- load: sahifa yuklanganda
- keydown/keyup: tugma bosilganda
- mouseover/mouseout: sichqoncha harakati

Event Listeners:
- addEventListener(): event listener qo'shish
- removeEventListener(): event listener olib tashlash
- Event object: event haqida ma'lumot

```
// DOM elementlarini tanlash
const tugma = document.getElementById('myButton');
const matn = document.querySelector('.text');
const barchaTugmalar = document.querySelectorAll('button');

// Element kontentini o'zgartirish
matn.innerHTML = '<strong>Yangi matn</strong>';
matn.textContent = 'Oddiy matn';
matn.style.color = 'blue';
matn.style.fontSize = '20px';

// Class'lar bilan ishlash
matn.classList.add('active');
matn.classList.remove('hidden');
matn.classList.toggle('highlight');

// Event listeners
tugma.addEventListener('click', function() {
    alert('Tugma bosildi!');
});

// Arrow function bilan
tugma.addEventListener('click', () => {
    console.log('Tugma bosildi!');
});

// Event object
document.addEventListener('keydown', function(event) {
    console.log('Bosilgan tugma: ' + event.key);
    if (event.key === 'Enter') {
        console.log('Enter bosildi!');
    }
});

// Form bilan ishlash
const forma = document.getElementById('myForm');
forma.addEventListener('submit', function(event) {
    event.preventDefault(); // Formaning standart yuborilishini to'xtatish

    const formData = new FormData(forma);
    const ism = formData.get('name');
    const email = formData.get('email');

    console.log('Ism: ' + ism);
    console.log('Email: ' + email);
});

// Dinamik element yaratish
function yangiElement() {
    const div = document.createElement('div');
    div.textContent = 'Yangi element';
    div.className = 'new-item';

    document.body.appendChild(div);
}

// Element o'chirish
function elementOchirish(element) {
    element.parentNode.removeChild(element);
}
```

Amaliy mashq:

To-do list yarating: yangi vazifa qo'shish, vazifani bajarish va o'chirish funksiyalari bilan.

5. React - Modern Frontend Framework

React asoslari va JSX

React - Facebook tomonidan yaratilgan JavaScript kutubxonasi.

React'ning afzalliklari:
- Component-based architecture
- Virtual DOM
- Unidirectional data flow
- Large ecosystem
- Strong community support

JSX (JavaScript XML):
- HTML'ga o'xshash sintaksis
- JavaScript ichida HTML yozish
- Babel tomonidan JavaScript'ga o'giriladi

React Components:
- Function Components (tavsiya etiladi)
- Class Components (eski usul)

Props:
- Parent'dan child'ga ma'lumot uzatish
- Read-only (o'zgartirib bo'lmaydi)
- Destructuring bilan ishlatish

State:
- Component'ning ichki holati
- useState Hook bilan boshqariladi
- State o'zgarishi component'ni qayta render qiladi

```
// React component yaratish
import React, { useState } from 'react';

// Function Component
function Salomlashish(props) {
    return <h1>Salom, {props.ism}!</h1>;
}

// Props bilan destructuring
function Foydalanuvchi({ ism, yosh, shahar }) {
    return (
        <div className="user-card">
            <h2>{ism}</h2>
            <p>Yosh: {yosh}</p>
            <p>Shahar: {shahar}</p>
        </div>
    );
}

// State bilan ishlash
function Hisoblagich() {
    const [son, setSon] = useState(0);

    const oshirish = () => {
        setSon(son + 1);
    };

    const kamaytirish = () => {
        setSon(son - 1);
    };

    return (
        <div className="counter">
            <h2>Hisoblagich: {son}</h2>
            <button onClick={oshirish}>+</button>
            <button onClick={kamaytirish}>-</button>
        </div>
    );
}

// Conditional rendering
function Xabar({ korinsin, matn }) {
    if (!korinsin) {
        return null;
    }

    return <div className="message">{matn}</div>;
}

// List rendering
function MevalarRoyxati() {
    const mevalar = ['olma', 'banan', 'apelsin', 'uzum'];

    return (
        <ul>
            {mevalar.map((meva, index) => (
                <li key={index}>{meva}</li>
            ))}
        </ul>
    );
}

// Main App component
function App() {
    const [xabarKorinsin, setXabarKorinsin] = useState(true);

    return (
        <div className="App">
            <Salomlashish ism="Ahmad" />
            <Foydalanuvchi
                ism="Ali"
                yosh={25}
                shahar="Toshkent"
            />
            <Hisoblagich />
            <Xabar
                korinsin={xabarKorinsin}
                matn="Bu muhim xabar!"
            />
            <button onClick={() => setXabarKorinsin(!xabarKorinsin)}>
                Xabarni {xabarKorinsin ? 'yashirish' : 'ko'rsatish'}
            </button>
            <MevalarRoyxati />
        </div>
    );
}

export default App;
```

Amaliy mashq:

Foydalanuvchi ma'lumotlarini ko'rsatadigan va tahrirlash imkonini beruvchi component yarating.

React Hooks va State Management

React Hooks - function component'larda state va lifecycle'dan foydalanish.

useState Hook:
- Component state'ini boshqarish
- [state, setState] qaytaradi
- Functional updates

useEffect Hook:
- Side effects va lifecycle events
- componentDidMount, componentDidUpdate, componentWillUnmount
- Dependency array
- Cleanup function

useContext Hook:
- Context API'dan foydalanish
- Prop drilling muammosini hal qilish
- Global state management

Custom Hooks:
- Logic'ni qayta ishlatish
- Hook'larni birlashtirish
- Clean code

State Management:
- Local state vs Global state
- Context API
- Redux (katta ilovalar uchun)
- Zustand (yengil alternativa)

```
import React, { useState, useEffect, useContext, createContext } from 'react';

// Context yaratish
const ThemeContext = createContext();

// Custom Hook
function useLocalStorage(key, initialValue) {
    const [storedValue, setStoredValue] = useState(() => {
        try {
            const item = window.localStorage.getItem(key);
            return item ? JSON.parse(item) : initialValue;
        } catch (error) {
            return initialValue;
        }
    });

    const setValue = (value) => {
        try {
            setStoredValue(value);
            window.localStorage.setItem(key, JSON.stringify(value));
        } catch (error) {
            console.error(error);
        }
    };

    return [storedValue, setValue];
}

// useEffect misoli
function MalumotYuklash() {
    const [malumot, setMalumot] = useState([]);
    const [yuklash, setYuklash] = useState(true);
    const [xato, setXato] = useState(null);

    useEffect(() => {
        async function fetchData() {
            try {
                setYuklash(true);
                const response = await fetch('/api/malumotlar');
                const data = await response.json();
                setMalumot(data);
            } catch (error) {
                setXato(error.message);
            } finally {
                setYuklash(false);
            }
        }

        fetchData();
    }, []); // Bo'sh dependency array - faqat mount'da ishga tushadi

    if (yuklash) return <div>Yuklanmoqda...</div>;
    if (xato) return <div>Xato: {xato}</div>;

    return (
        <ul>
            {malumot.map(item => (
                <li key={item.id}>{item.name}</li>
            ))}
        </ul>
    );
}

// Context Provider
function ThemeProvider({ children }) {
    const [theme, setTheme] = useLocalStorage('theme', 'light');

    const toggleTheme = () => {
        setTheme(theme === 'light' ? 'dark' : 'light');
    };

    return (
        <ThemeContext.Provider value={{ theme, toggleTheme }}>
            {children}
        </ThemeContext.Provider>
    );
}

// Context Consumer
function ThemeToggle() {
    const { theme, toggleTheme } = useContext(ThemeContext);

    return (
        <button onClick={toggleTheme}>
                         { t h e m e   = = =   ' l i g h t '   ?   'Ø<ß  '   :   '& þ  ' }
            {theme === 'light' ? 'Qorong'u' : 'Yorug''}
        </button>
    );
```

```
}

// Timer Hook misoli
function useTimer(initialTime = 0) {
    const [time, setTime] = useState(initialTime);
    const [isRunning, setIsRunning] = useState(false);

    useEffect(() => {
        let interval = null;

        if (isRunning) {
            interval = setInterval(() => {
                setTime(time => time + 1);
            }, 1000);
        } else if (!isRunning && time !== 0) {
            clearInterval(interval);
        }

        return () => clearInterval(interval);
    }, [isRunning, time]);

    const start = () => setIsRunning(true);
    const stop = () => setIsRunning(false);
    const reset = () => {
        setTime(0);
        setIsRunning(false);
    };

    return { time, isRunning, start, stop, reset };
}

// Timer component
function Timer() {
    const { time, isRunning, start, stop, reset } = useTimer();

    return (
        <div>
            <h2>Vaqt: {time} soniya</h2>
            <button onClick={start} disabled={isRunning}>Boshlash</button>
            <button onClick={stop} disabled={!isRunning}>To'xtatish</button>
            <button onClick={reset}>Qayta boshlash</button>
        </div>
    );
}
```

Amaliy mashq:

useLocalStorage custom hook'idan foydalanib, foydalanuvchi sozlamalarini saqlaydigan component yarating.

6. Node.js va Backend Development

Node.js va Express.js asoslari

Node.js - server tomonida JavaScript ishlatish imkonini beradi.

Node.js'ning afzalliklari:
- JavaScript everywhere
- Non-blocking I/O
- Large ecosystem (npm)
- Fast development
- Scalable applications

Express.js - Node.js uchun minimal web framework:
- Routing
- Middleware
- Template engines
- Static files serving
- Error handling

HTTP metodlari:
- GET: ma'lumot olish
- POST: yangi ma'lumot yaratish
- PUT: ma'lumotni yangilash
- DELETE: ma'lumotni o'chirish
- PATCH: qisman yangilash

Middleware:
- Request va response orasidagi funksiyalar
- Authentication, logging, parsing
- Error handling
- Custom middleware

```
// server.js
const express = require('express');
const cors = require('cors');
const morgan = require('morgan');

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(cors()); // CORS ni yoqish
app.use(express.json()); // JSON parsing
app.use(express.urlencoded({ extended: true })); // URL encoding
app.use(morgan('combined')); // Logging

// Static files
app.use(express.static('public'));

// Routes
app.get('/', (req, res) => {
    res.json({
        xabar: 'Salom, bu mening API serverim!',
        vaqt: new Date().toISOString()
    });
});
```

```javascript
// Ma'lumotlar (haqiqiy loyihada database ishlatiladi)
let foydalanuvchilar = [
    { id: 1, ism: 'Ali', email: 'ali@example.com' },
    { id: 2, ism: 'Oyna', email: 'oyna@example.com' },
    { id: 3, ism: 'Bobur', email: 'bobur@example.com' }
];

// GET - Barcha foydalanuvchilar
app.get('/api/foydalanuvchilar', (req, res) => {
    res.json(foydalanuvchilar);
});

// GET - Bitta foydalanuvchi
app.get('/api/foydalanuvchilar/:id', (req, res) => {
    const id = parseInt(req.params.id);
    const foydalanuvchi = foydalanuvchilar.find(f => f.id === id);

    if (!foydalanuvchi) {
        return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
    }

    res.json(foydalanuvchi);
});

// POST - Yangi foydalanuvchi qo'shish
app.post('/api/foydalanuvchilar', (req, res) => {
    const { ism, email } = req.body;

    // Validation
    if (!ism || !email) {
        return res.status(400).json({
            xato: 'Ism va email talab qilinadi'
        });
    }

    const yangiFoydalanuvchi = {
        id: foydalanuvchilar.length + 1,
        ism,
        email
    };

    foydalanuvchilar.push(yangiFoydalanuvchi);
    res.status(201).json(yangiFoydalanuvchi);
});

// PUT - Foydalanuvchini yangilash
app.put('/api/foydalanuvchilar/:id', (req, res) => {
    const id = parseInt(req.params.id);
    const foydalanuvchiIndex = foydalanuvchilar.findIndex(f => f.id === id);

    if (foydalanuvchiIndex === -1) {
        return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
    }

    const { ism, email } = req.body;
    foydalanuvchilar[foydalanuvchiIndex] = { id, ism, email };

    res.json(foydalanuvchilar[foydalanuvchiIndex]);
});

// DELETE - Foydalanuvchini o'chirish
app.delete('/api/foydalanuvchilar/:id', (req, res) => {
    const id = parseInt(req.params.id);
    const foydalanuvchiIndex = foydalanuvchilar.findIndex(f => f.id === id);
    if (foydalanuvchiIndex === -1) {
        return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
    }

    foydalanuvchilar.splice(foydalanuvchiIndex, 1);
    res.json({ xabar: 'Foydalanuvchi o'chirildi' });
});

// Error handling middleware
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).json({ xato: 'Server xatosi yuz berdi' });
});

// 404 handler
app.use((req, res) => {
    res.status(404).json({ xato: 'Sahifa topilmadi' });
});

// Serverni ishga tushirish
app.listen(PORT, () => {
    console.log(`Server ${PORT} portida ishlamoqda`);
});
```

Amaliy mashq:

Mahsulotlar uchun CRUD API yarating: mahsulot qo'shish, o'qish, yangilash va o'chirish.

Ma'lumotlar bazasi - MongoDB

MongoDB - mashhur NoSQL ma'lumotlar bazasi.

MongoDB'ning afzalliklari:
- Document-based (JSON-like)
- Flexible schema
- Horizontal scaling
- Rich query language
- Aggregation framework

Mongoose - MongoDB uchun ODM (Object Document Mapper):
- Schema definition
- Model creation
- Validation
- Middleware
- Population

CRUD operatsiyalar:
- Create: save(), create()
- Read: find(), findOne(), findById()
- Update: updateOne(), findByIdAndUpdate()
- Delete: deleteOne(), findByIdAndDelete()

Schema types:
- String, Number, Date, Boolean
- Array, Mixed, ObjectId
- Custom validation

- Default values

```
// MongoDB bilan ishlash - Mongoose
const mongoose = require('mongoose');

// Ma'lumotlar bazasiga ulanish
mongoose.connect('mongodb://localhost:27017/mening_ilovam', {
    useNewUrlParser: true,
    useUnifiedTopology: true
});

// Schema yaratish
const FoydalanuvchiSchema = new mongoose.Schema({
    ism: {
        type: String,
        required: [true, 'Ism talab qilinadi'],
        trim: true,
        minlength: [2, 'Ism kamida 2 ta harf bo'lishi kerak'],
        maxlength: [50, 'Ism 50 ta harfdan oshmasligi kerak']
    },
    email: {
        type: String,
        required: [true, 'Email talab qilinadi'],
        unique: true,
        lowercase: true,
        match: [/^w+([.-]?w+)*@w+([.-]?w+)*(.w{2,3})+$/, 'Email formati noto'g'ri']
    },
    yosh: {
        type: Number,
        min: [0, 'Yosh manfiy bo'lishi mumkin emas'],
        max: [120, 'Yosh 120 dan oshmasligi kerak']
    },
    shahar: {
        type: String,
        default: 'Toshkent'
    },
    faol: {
        type: Boolean,
        default: true
    },
    yaratilganVaqt: {
        type: Date,
        default: Date.now
    },
    oxirgiKirish: Date
});

// Virtual field
FoydalanuvchiSchema.virtual('toliqIsm').get(function() {
    return this.ism + ' (' + this.email + ')';
});

// Pre middleware
FoydalanuvchiSchema.pre('save', function(next) {
    console.log('Foydalanuvchi saqlanmoqda:', this.ism);
    next();
});

// Model yaratish
const Foydalanuvchi = mongoose.model('Foydalanuvchi', FoydalanuvchiSchema);

// Express route'larda ishlatish
const express = require('express');
const app = express();

app.use(express.json());
```

```
// Yangi foydalanuvchi yaratish
app.post('/api/foydalanuvchilar', async (req, res) => {
    try {
        const yangiFoydalanuvchi = new Foydalanuvchi(req.body);
        await yangiFoydalanuvchi.save();

        res.status(201).json(yangiFoydalanuvchi);
    } catch (xato) {
        if (xato.name === 'ValidationError') {
            const xatolar = Object.values(xato.errors).map(err => err.message);
            return res.status(400).json({ xato: xatolar });
        }

        if (xato.code === 11000) {
            return res.status(409).json({
                xato: 'Bu email allaqachon ishlatilgan'
            });
        }

        res.status(500).json({ xato: 'Server xatosi' });
    }
});

// Barcha foydalanuvchilarni olish
app.get('/api/foydalanuvchilar', async (req, res) => {
    try {
        const { sahifa = 1, limit = 10, qidiruv } = req.query;

        let filter = { faol: true };
        if (qidiruv) {
            filter.$or = [
                { ism: { $regex: qidiruv, $options: 'i' } },
                { email: { $regex: qidiruv, $options: 'i' } }
            ];
        }

        const foydalanuvchilar = await Foydalanuvchi
            .find(filter)
            .limit(limit * 1)
            .skip((sahifa - 1) * limit)
            .sort({ yaratilganVaqt: -1 });

        const jami = await Foydalanuvchi.countDocuments(filter);

        res.json({
            foydalanuvchilar,
            sahifaInfo: {
                joriy: parseInt(sahifa),
                jami: Math.ceil(jami / limit),
                elementlarSoni: jami
            }
        });
    } catch (xato) {
        res.status(500).json({ xato: xato.message });
    }
});

// Bitta foydalanuvchini olish
app.get('/api/foydalanuvchilar/:id', async (req, res) => {
    try {
        const foydalanuvchi = await Foydalanuvchi.findById(req.params.id);

        if (!foydalanuvchi) {
            return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
        }
        res.json(foydalanuvchi);
    } catch (xato) {
        res.status(500).json({ xato: xato.message });
    }
});

// Foydalanuvchini yangilash
app.put('/api/foydalanuvchilar/:id', async (req, res) => {
    try {
        const foydalanuvchi = await Foydalanuvchi.findByIdAndUpdate(
            req.params.id,
            req.body,
            { new: true, runValidators: true }
        );

        if (!foydalanuvchi) {
            return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
        }

        res.json(foydalanuvchi);
    } catch (xato) {
        res.status(400).json({ xato: xato.message });
    }
});

// Foydalanuvchini o'chirish (soft delete)
app.delete('/api/foydalanuvchilar/:id', async (req, res) => {
    try {
        const foydalanuvchi = await Foydalanuvchi.findByIdAndUpdate(
            req.params.id,
            { faol: false },
            { new: true }
        );

        if (!foydalanuvchi) {
            return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
        }

        res.json({ xabar: 'Foydalanuvchi o'chirildi' });
    } catch (xato) {
        res.status(500).json({ xato: xato.message });
    }
});
```

Amaliy mashq:

Blog post'lari uchun MongoDB schema yarating va CRUD operatsiyalarini amalga oshiring.

7. Authentication va Authorization

JWT va Session-based Authentication

Authentication - foydalanuvchini tanib olish jarayoni.
Authorization - foydalanuvchiga ruxsat berish jarayoni.

Authentication turlari:
1. Session-based: server'da session saqlanadi
2. Token-based: JWT (JSON Web Token) ishlatiladi

JWT (JSON Web Token):
- Header: algoritm va token turi
- Payload: foydalanuvchi ma'lumotlari
- Signature: xavfsizlik uchun imzo

JWT'ning afzalliklari:
- Stateless
- Scalable
- Cross-domain
- Mobile-friendly

Password Security:
- Hashing: bcrypt, scrypt, argon2
- Salt: random data
- Pepper: server secret
- Password strength validation

Session vs JWT:
- Session: server memory, database
- JWT: client storage, no server state

```
// Authentication middleware
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const User = require('../models/User');

// JWT secret key (environment variable'da saqlash kerak)
const JWT_SECRET = process.env.JWT_SECRET || 'supersecretkey';

// Password hash qilish
async function hashPassword(password) {
    const saltRounds = 12;
    return await bcrypt.hash(password, saltRounds);
}

// Password tekshirish
async function comparePassword(password, hashedPassword) {
    return await bcrypt.compare(password, hashedPassword);
}

// JWT token yaratish
function generateToken(userId) {
    return jwt.sign(
        { userId },
        JWT_SECRET,
        { expiresIn: '7d' }
    );
}

// JWT token tekshirish middleware
function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

    if (!token) {
        return res.status(401).json({ xato: 'Token talab qilinadi' });
    }

    jwt.verify(token, JWT_SECRET, (err, decoded) => {
        if (err) {
            return res.status(403).json({ xato: 'Token noto'g'ri yoki muddati tugagan' });
        }

        req.userId = decoded.userId;
        next();
    });
}

// Ro'yxatdan o'tish
app.post('/api/auth/register', async (req, res) => {
    try {
        const { ism, email, parol } = req.body;

        // Validation
        if (!ism || !email || !parol) {
            return res.status(400).json({
                xato: 'Barcha maydonlar talab qilinadi'
            });
        }

        if (parol.length < 6) {
            return res.status(400).json({
                xato: 'Parol kamida 6 ta belgidan iborat bo'lishi kerak'
            });
        }

        // Foydalanuvchi mavjudligini tekshirish
        const mavjudFoydalanuvchi = await User.findOne({ email });
        if (mavjudFoydalanuvchi) {
            return res.status(409).json({
                xato: 'Bu email allaqachon ishlatilgan'
            });
        }

        // Parolni hash qilish
        const hashedPassword = await hashPassword(parol);

        // Yangi foydalanuvchi yaratish
        const yangiFoydalanuvchi = new User({
            ism,
            email,
            parol: hashedPassword
        });

        await yangiFoydalanuvchi.save();

        // Token yaratish
        const token = generateToken(yangiFoydalanuvchi._id);

        // Parolni javobdan olib tashlash
        const { parol: _, ...foydalanuvchiData } = yangiFoydalanuvchi.toObject();
        res.status(201).json({
            xabar: 'Foydalanuvchi muvaffaqiyatli ro'yxatdan o'tdi',
            foydalanuvchi: foydalanuvchiData,
            token
        });

    } catch (xato) {
        console.error(xato);
        res.status(500).json({ xato: 'Server xatosi yuz berdi' });
    }
});

// Kirish
app.post('/api/auth/login', async (req, res) => {
    try {
        const { email, parol } = req.body;

        // Validation
        if (!email || !parol) {
            return res.status(400).json({
                xato: 'Email va parol talab qilinadi'
            });
        }

        // Foydalanuvchini topish
        const foydalanuvchi = await User.findOne({ email });
        if (!foydalanuvchi) {
            return res.status(401).json({
                xato: 'Email yoki parol noto'g'ri'
            });
        }

        // Parolni tekshirish
        const parolTogri = await comparePassword(parol, foydalanuvchi.parol);
        if (!parolTogri) {
            return res.status(401).json({
                xato: 'Email yoki parol noto'g'ri'
            });
        }

        // Oxirgi kirish vaqtini yangilash
        foydalanuvchi.oxirgiKirish = new Date();
        await foydalanuvchi.save();

        // Token yaratish
        const token = generateToken(foydalanuvchi._id);

        // Parolni javobdan olib tashlash
        const { parol: _, ...foydalanuvchiData } = foydalanuvchi.toObject();

        res.json({
            xabar: 'Muvaffaqiyatli kirildi',
            foydalanuvchi: foydalanuvchiData,
            token
        });

    } catch (xato) {
        console.error(xato);
        res.status(500).json({ xato: 'Server xatosi yuz berdi' });
    }
});

// Himoyalangan route
app.get('/api/profile', authenticateToken, async (req, res) => {
    try {
        const foydalanuvchi = await User.findById(req.userId).select('-parol');
        if (!foydalanuvchi) {
            return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
        }

        res.json(foydalanuvchi);
    } catch (xato) {
        res.status(500).json({ xato: 'Server xatosi' });
    }
});

// Role-based authorization
function authorize(roles = []) {
    return async (req, res, next) => {
        try {
            const foydalanuvchi = await User.findById(req.userId);

            if (!foydalanuvchi) {
                return res.status(404).json({ xato: 'Foydalanuvchi topilmadi' });
            }

            if (roles.length && !roles.includes(foydalanuvchi.rol)) {
                return res.status(403).json({
                    xato: 'Bu amalni bajarish uchun ruxsat yo'q'
                });
            }

            next();
        } catch (xato) {
            res.status(500).json({ xato: 'Server xatosi' });
        }
    };
}

// Admin faqat route
app.get('/api/admin/users',
    authenticateToken,
    authorize(['admin']),
    async (req, res) => {
        // Admin faqat foydalanuvchilar ro'yxatini ko'rishi mumkin
        const users = await User.find().select('-parol');
        res.json(users);
    }
);
```

Amaliy mashq:

Foydalanuvchi ro'yxatdan o'tish va kirish sistemasini yarating, JWT token bilan himoyalangan route'lar qo'shing.

8. API Development va Testing

RESTful API va Documentation

REST (Representational State Transfer) - API yaratish uchun arxitektura stili.

REST prinsiplan:
1. Stateless: har bir so'rov mustaqil
2. Client-Server: ajratilgan arxitektura
3. Cacheable: kesh qilish mumkin
4. Uniform Interface: bir xil interfeys
5. Layered System: qatlamli tizim

HTTP Status Codes:
- 200: OK - muvaffaqiyatli
- 201: Created - yaratildi
- 400: Bad Request - noto'g'ri so'rov
- 401: Unauthorized - autentifikatsiya kerak
- 403: Forbidden - ruxsat yo'q
- 404: Not Found - topilmadi
- 500: Internal Server Error - server xatosi

API Documentation:
- Swagger/OpenAPI
- Postman Collections
- API Blueprint
- Clear examples
- Error responses

API Versioning:
- URL versioning: /api/v1/users
- Header versioning: Accept: application/vnd.api+json;version=1
- Query parameter: /api/users?version=1

```
// RESTful API - To'liq misol
const express = require('express');
const mongoose = require('mongoose');
const rateLimit = require('express-rate-limit');
const helmet = require('helmet');
const cors = require('cors');

const app = express();

// Security middleware
app.use(helmet());
app.use(cors());

// Rate limiting
const limiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 daqiqa
    max: 100, // maksimal 100 ta so'rov
    message: 'Juda ko'p so'rov yuborildi, keyinroq urinib ko'ring'
});
app.use('/api/', limiter);

app.use(express.json({ limit: '10mb' }));
```

```javascript
// Mahsulot modeli
const MahsulotSchema = new mongoose.Schema({
    nom: {
        type: String,
        required: true,
        trim: true,
        maxlength: 100
    },
    tavsif: {
        type: String,
        maxlength: 500
    },
    narx: {
        type: Number,
        required: true,
        min: 0
    },
    kategoriya: {
        type: String,
        required: true,
        enum: ['elektronika', 'kiyim', 'kitob', 'sport', 'boshqa']
    },
    stokMiqdor: {
        type: Number,
        default: 0,
        min: 0
    },
    rasm: String,
    faol: {
        type: Boolean,
        default: true
    },
    yaratilganVaqt: {
        type: Date,
        default: Date.now
    },
    yangilanganVaqt: {
        type: Date,
        default: Date.now
    }
});

// Pre-save middleware
MahsulotSchema.pre('save', function(next) {
    this.yangilanganVaqt = Date.now();
    next();
});

const Mahsulot = mongoose.model('Mahsulot', MahsulotSchema);

// API Routes

// GET /api/mahsulotlar - Barcha mahsulotlar (pagination, filtering, sorting)
app.get('/api/mahsulotlar', async (req, res) => {
    try {
        const {
            sahifa = 1,
            limit = 10,
            kategoriya,
            qidiruv,
            minNarx,
            maxNarx,
            sort = 'yaratilganVaqt'
        } = req.query;
        // Filter yaratish
        let filter = { faol: true };

        if (kategoriya) {
            filter.kategoriya = kategoriya;
        }

        if (qidiruv) {
            filter.$or = [
                { nom: { $regex: qidiruv, $options: 'i' } },
                { tavsif: { $regex: qidiruv, $options: 'i' } }
            ];
        }

        if (minNarx || maxNarx) {
            filter.narx = {};
            if (minNarx) filter.narx.$gte = parseFloat(minNarx);
            if (maxNarx) filter.narx.$lte = parseFloat(maxNarx);
        }

        // Sorting
        let sortObj = {};
        if (sort.startsWith('-')) {
            sortObj[sort.substring(1)] = -1;
        } else {
            sortObj[sort] = 1;
        }

        // Ma'lumotlarni olish
        const mahsulotlar = await Mahsulot
            .find(filter)
            .sort(sortObj)
            .limit(limit * 1)
            .skip((sahifa - 1) * limit);

        const jami = await Mahsulot.countDocuments(filter);

        res.json({
            success: true,
            data: mahsulotlar,
            pagination: {
                joriy: parseInt(sahifa),
                jami: Math.ceil(jami / limit),
                elementlarSoni: jami,
                limit: parseInt(limit)
            }
        });

    } catch (xato) {
        res.status(500).json({
            success: false,
            xato: 'Mahsulotlarni olishda xato yuz berdi'
        });
    }
});

// GET /api/mahsulotlar/:id - Bitta mahsulot
app.get('/api/mahsulotlar/:id', async (req, res) => {
    try {
        const mahsulot = await Mahsulot.findById(req.params.id);

        if (!mahsulot || !mahsulot.faol) {
            return res.status(404).json({
                success: false,
                xato: 'Mahsulot topilmadi'
            });
        }

        res.json({
            success: true,
            data: mahsulot
        });

    } catch (xato) {
        if (xato.name === 'CastError') {
            return res.status(400).json({
                success: false,
                xato: 'Noto'g'ri mahsulot ID'
            });
        }

        res.status(500).json({
            success: false,
            xato: 'Server xatosi'
        });
    }
});

// POST /api/mahsulotlar - Yangi mahsulot yaratish
app.post('/api/mahsulotlar', authenticateToken, async (req, res) => {
    try {
        const yangimahsulot = new Mahsulot(req.body);
        await yangimahsulot.save();

        res.status(201).json({
            success: true,
            data: yangimahsulot,
            xabar: 'Mahsulot muvaffaqiyatli yaratildi'
        });

    } catch (xato) {
        if (xato.name === 'ValidationError') {
            const xatolar = Object.values(xato.errors).map(err => err.message);
            return res.status(400).json({
                success: false,
                xato: 'Validation xatosi',
                details: xatolar
            });
        }

        res.status(500).json({
            success: false,
            xato: 'Mahsulot yaratishda xato yuz berdi'
        });
    }
});

// PUT /api/mahsulotlar/:id - Mahsulotni yangilash
app.put('/api/mahsulotlar/:id', authenticateToken, async (req, res) => {
    try {
        const mahsulot = await Mahsulot.findByIdAndUpdate(
            req.params.id,
            { ...req.body, yangilanganVaqt: Date.now() },
            { new: true, runValidators: true }
        );

        if (!mahsulot) {
            return res.status(404).json({
                success: false,
                xato: 'Mahsulot topilmadi'
            });
        }
        res.json({
            success: true,
            data: mahsulot,
            xabar: 'Mahsulot muvaffaqiyatli yangilandi'
        });

    } catch (xato) {
        if (xato.name === 'ValidationError') {
            const xatolar = Object.values(xato.errors).map(err => err.message);
            return res.status(400).json({
                success: false,
                xato: 'Validation xatosi',
                details: xatolar
            });
        }

        res.status(500).json({
            success: false,
            xato: 'Mahsulotni yangilashda xato yuz berdi'
        });
    }
});

// DELETE /api/mahsulotlar/:id - Mahsulotni o'chirish (soft delete)
app.delete('/api/mahsulotlar/:id', authenticateToken, async (req, res) => {
    try {
        const mahsulot = await Mahsulot.findByIdAndUpdate(
            req.params.id,
            { faol: false, yangilanganVaqt: Date.now() },
            { new: true }
        );

        if (!mahsulot) {
            return res.status(404).json({
                success: false,
                xato: 'Mahsulot topilmadi'
            });
        }

        res.json({
            success: true,
            xabar: 'Mahsulot muvaffaqiyatli o'chirildi'
        });

    } catch (xato) {
        res.status(500).json({
            success: false,
            xato: 'Mahsulotni o'chirishda xato yuz berdi'
        });
    }
});

// Global error handler
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).json({
        success: false,
        xato: 'Server xatosi yuz berdi'
    });
});

// 404 handler
app.use('*', (req, res) => {
    res.status(404).json({
        success: false,
        xato: 'API endpoint topilmadi'
    });
});
```

Amaliy mashq:

Blog API yarating: post'lar, kommentlar va kategoriyalar uchun to'liq CRUD operatsiyalar bilan.

9. Frontend va Backend Integration

API Integration va State Management

Frontend va Backend orasidagi bog'lanish - full-stack development'ning muhim qismi.

API Integration:
- Fetch API
- Axios library
- Error handling
- Loading states
- Caching strategies

State Management:
- Local state (useState)
- Global state (Context API, Redux)
- Server state (React Query, SWR)
- Form state (Formik, React Hook Form)

Data Fetching Patterns:
- Fetch on mount
- Fetch on demand
- Optimistic updates
- Background refetching
- Infinite scrolling

Error Handling:
- Network errors
- Server errors
- Validation errors
- User-friendly messages
- Retry mechanisms

```
// API service layer
class ApiService {
    constructor(baseURL = '/api') {
        this.baseURL = baseURL;
        this.token = localStorage.getItem('token');
    }

    // Request interceptor
    async request(endpoint, options = {}) {
        const url = `${this.baseURL}${endpoint}`;

        const config = {
            headers: {
                'Content-Type': 'application/json',
                ...options.headers,
            },
            ...options,
        };

        if (this.token) {
            config.headers.Authorization = `Bearer ${this.token}`;
        }

        try {
            const response = await fetch(url, config);

            if (!response.ok) {
                const errorData = await response.json();
                throw new Error(errorData.xato || 'Network error');
            }

            return await response.json();
        } catch (error) {
            console.error('API Error:', error);
            throw error;
        }
    }

    // HTTP methods
    get(endpoint) {
        return this.request(endpoint);
    }

    post(endpoint, data) {
        return this.request(endpoint, {
            method: 'POST',
            body: JSON.stringify(data),
        });
    }

    put(endpoint, data) {
        return this.request(endpoint, {
            method: 'PUT',
            body: JSON.stringify(data),
        });
    }

    delete(endpoint) {
        return this.request(endpoint, {
            method: 'DELETE',
        });
    }

    // Authentication methods
    setToken(token) {
        this.token = token;
        localStorage.setItem('token', token);
    }

    clearToken() {
        this.token = null;
        localStorage.removeItem('token');
    }
}

const api = new ApiService();

// React hooks for API calls
import React, { useState, useEffect, useContext, createContext } from 'react';

// Auth Context
const AuthContext = createContext();

export function AuthProvider({ children }) {
    const [user, setUser] = useState(null);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const token = localStorage.getItem('token');
        if (token) {
            api.setToken(token);
            fetchUser();
        } else {
            setLoading(false);
        }
    }, []);

    const fetchUser = async () => {
        try {
            const userData = await api.get('/auth/profile');
            setUser(userData.data);
        } catch (error) {
            console.error('User fetch error:', error);
            api.clearToken();
        } finally {
            setLoading(false);
        }
    };

    const login = async (email, parol) => {
        try {
            const response = await api.post('/auth/login', { email, parol });
            const { token, foydalanuvchi } = response;

            api.setToken(token);
            setUser(foydalanuvchi);

            return { success: true };
        } catch (error) {
            return { success: false, error: error.message };
        }
    };

    const logout = () => {
        api.clearToken();
        setUser(null);
    };

    const value = {
        user,
        login,
        logout,
        loading
    };

    return (
        <AuthContext.Provider value={value}>
            {children}
        </AuthContext.Provider>
    );
}

export const useAuth = () => {
    const context = useContext(AuthContext);
    if (!context) {
        throw new Error('useAuth must be used within AuthProvider');
    }
    return context;
};

// Custom hook for data fetching
function useApi(endpoint, dependencies = []) {
    const [data, setData] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
    useEffect(() => {
        let cancelled = false;

        const fetchData = async () => {
            try {
                setLoading(true);
                setError(null);

                const result = await api.get(endpoint);

                if (!cancelled) {
                    setData(result.data);
                }
            } catch (err) {
                if (!cancelled) {
                    setError(err.message);
                }
            } finally {
                if (!cancelled) {
                    setLoading(false);
                }
            }
        };

        fetchData();

        return () => {
            cancelled = true;
        };
    }, dependencies);

    const refetch = () => {
        setLoading(true);
        setError(null);
        // Trigger useEffect
    };

    return { data, loading, error, refetch };
}

// Mahsulotlar komponenti
function MahsulotlarRoyxati() {
    const [sahifa, setSahifa] = useState(1);
    const [qidiruv, setQidiruv] = useState('');
    const [kategoriya, setKategoriya] = useState('');

    const { data, loading, error, refetch } = useApi(
        `/mahsulotlar?sahifa=${sahifa}&qidiruv=${qidiruv}&kategoriya=${kategoriya}`,
        [sahifa, qidiruv, kategoriya]
    );

    const handleSearch = (e) => {
        e.preventDefault();
        setSahifa(1); // Reset to first page
        refetch();
    };

    if (loading) return <div className="loading">Yuklanmoqda...</div>;
    if (error) return <div className="error">Xato: {error}</div>;

    return (
        <div className="mahsulotlar-container">
            <div className="search-form">
                <form onSubmit={handleSearch}>
                    <input
                        type="text"
                        placeholder="Mahsulot qidirish..."
                        value={qidiruv}
                        onChange={(e) => setQidiruv(e.target.value)}
                    />
                    <select
                        value={kategoriya}
                        onChange={(e) => setKategoriya(e.target.value)}
                    >
                        <option value="">Barcha kategoriyalar</option>
                        <option value="elektronika">Elektronika</option>
                        <option value="kiyim">Kiyim</option>
                        <option value="kitob">Kitob</option>
                    </select>
                    <button type="submit">Qidirish</button>
                </form>
            </div>

            <div className="mahsulotlar-grid">
                {data?.mahsulotlar?.map(mahsulot => (
                    <MahsulotKarti key={mahsulot._id} mahsulot={mahsulot} />
                ))}
            </div>

            {data?.pagination && (
                <Pagination
                    joriy={data.pagination.joriy}
                    jami={data.pagination.jami}
                    onChange={setSahifa}
                />
            )}
        </div>
    );
}

// Mahsulot kartasi komponenti
function MahsulotKarti({ mahsulot }) {
    const [loading, setLoading] = useState(false);

    const handleAddToCart = async () => {
        setLoading(true);
        try {
            await api.post('/cart/add', {
                mahsulotId: mahsulot._id,
                miqdor: 1
            });
            // Success notification
        } catch (error) {
            // Error notification
        } finally {
            setLoading(false);
        }
    };

    return (
        <div className="mahsulot-karti">
            <img src={mahsulot.rasm} alt={mahsulot.nom} />
            <h3>{mahsulot.nom}</h3>
            <p className="narx">{mahsulot.narx.toLocaleString()} so'm</p>
            <p className="tavsif">{mahsulot.tavsif}</p>
            <button
                onClick={handleAddToCart}
                disabled={loading || mahsulot.stokMiqdor === 0}
                className="add-to-cart-btn"
            >
                {loading ? 'Qo'shilmoqda...' : 'Savatga qo'shish'}
            </button>
        </div>
    );
}
```

Amaliy mashq:

To'liq e-commerce frontend yarating: mahsulotlar ro'yxati, qidiruv, filtrlash va savatga qo'shish funksiyalari bilan.

10. Deployment va Production

Production Environment va Deployment

Production ga chiqarish - development jarayonining so'nggi bosqichi.

Production Environment:
- Environment variables
- Database configuration
- Security settings
- Performance optimization
- Monitoring va logging

Deployment Platforms:
Frontend:
- Netlify
- Vercel
- GitHub Pages
- AWS S3 + CloudFront

Backend:
- Heroku
- DigitalOcean
- AWS EC2
- Google Cloud Platform

Database:
- MongoDB Atlas
- AWS RDS
- PostgreSQL on Heroku
- Firebase Firestore

CI/CD (Continuous Integration/Deployment):
- GitHub Actions
- GitLab CI/CD
- Jenkins
- Travis CI

Performance Optimization:
- Code splitting
- Lazy loading
- Image optimization
- Caching strategies
- CDN usage

```
// package.json - Build scripts
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "start": "node server.js",
    "test": "jest",
    "lint": "eslint src/",
    "deploy": "npm run build && npm run deploy:netlify"
  },
  "engines": {
    "node": ">=16.0.0"
  }
}

// Environment variables - .env.production
NODE_ENV=production
DATABASE_URL=mongodb+srv://user:password@cluster.mongodb.net/myapp
JWT_SECRET=supersecretproductionkey
PORT=3000
FRONTEND_URL=https://myapp.netlify.app

// server.js - Production configuration
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const helmet = require('helmet');
const compression = require('compression');
const rateLimit = require('express-rate-limit');

const app = express();
const PORT = process.env.PORT || 3000;

// Security middleware
app.use(helmet({
    contentSecurityPolicy: {
        directives: {
            defaultSrc: ["'self'"],
            styleSrc: ["'self'", "'unsafe-inline'"],
            scriptSrc: ["'self'"],
            imgSrc: ["'self'", "data:", "https:"],
        },
    },
}));

// CORS configuration
app.use(cors({
    origin: process.env.FRONTEND_URL || 'http://localhost:3000',
    credentials: true
}));

// Compression
app.use(compression());

// Rate limiting
const limiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: process.env.NODE_ENV === 'production' ? 100 : 1000,
    message: 'Too many requests from this IP'
});
app.use('/api/', limiter);

// Body parsing
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));
```

```
// Static files (for production)
if (process.env.NODE_ENV === 'production') {
    app.use(express.static('dist'));

    app.get('*', (req, res) => {
        res.sendFile(path.join(__dirname, 'dist', 'index.html'));
    });
}

// Database connection
mongoose.connect(process.env.DATABASE_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
})
.then(() => console.log('Database connected'))
.catch(err => console.error('Database connection error:', err));

// Health check endpoint
app.get('/health', (req, res) => {
    res.json({
        status: 'OK',
        timestamp: new Date().toISOString(),
        uptime: process.uptime()
    });
});

// Error handling
app.use((err, req, res, next) => {
    console.error(err.stack);

    if (process.env.NODE_ENV === 'production') {
        res.status(500).json({ error: 'Something went wrong!' });
    } else {
        res.status(500).json({ error: err.message, stack: err.stack });
    }
});

// Graceful shutdown
process.on('SIGTERM', () => {
    console.log('SIGTERM received, shutting down gracefully');
    mongoose.connection.close(() => {
        console.log('Database connection closed');
        process.exit(0);
    });
});

app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});

// Dockerfile
FROM node:16-alpine

WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci --only=production

# Copy source code
COPY . .

# Build frontend (if applicable)
```

```
RUN npm run build

# Expose port
EXPOSE 3000

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD curl -f http://localhost:3000/health || exit 1

# Start application
CMD ["npm", "start"]

// docker-compose.yml
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=mongodb://mongo:27017/myapp
      - JWT_SECRET=supersecretkey
    depends_on:
      - mongo
    restart: unless-stopped

  mongo:
    image: mongo:5.0
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
    restart: unless-stopped

volumes:
  mongo_data:

// GitHub Actions - .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Setup Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '16'
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run tests
      run: npm test

    - name: Run linting
```

```
      run: npm run lint

  deploy:
    needs: test
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Setup Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '16'
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Build application
      run: npm run build

    - name: Deploy to Netlify
      uses: nwtgck/actions-netlify@v1.2
      with:
        publish-dir: './dist'
        production-branch: main
        github-token: ${{ secrets.GITHUB_TOKEN }}
        deploy-message: "Deploy from GitHub Actions"
      env:
        NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
        NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}

// Performance monitoring
const winston = require('winston');

const logger = winston.createLogger({
    level: 'info',
    format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.errors({ stack: true }),
        winston.format.json()
    ),
    defaultMeta: { service: 'myapp' },
    transports: [
        new winston.transports.File({ filename: 'error.log', level: 'error' }),
        new winston.transports.File({ filename: 'combined.log' })
    ]
});

if (process.env.NODE_ENV !== 'production') {
    logger.add(new winston.transports.Console({
        format: winston.format.simple()
    }));
}

// Request logging middleware
app.use((req, res, next) => {
    const start = Date.now();

    res.on('finish', () => {
        const duration = Date.now() - start;
        logger.info({
            method: req.method,
            url: req.url,
            status: res.statusCode,
            duration: `${duration}ms`,
            ip: req.ip,
            userAgent: req.get('User-Agent')
        });
    });

    next();
});
```

Amaliy mashq:

O'z loyihangizni production muhitiga deploy qiling: environment variables, security va monitoring bilan.

Xulosa

Tabriklaymiz! Siz Full-Stack Web Development bo'yicha to'liq kursni yakunladingiz.

Endi sizda quyidagi ko'nikmalar mavjud:

- HTML, CSS va JavaScript asoslari
- React bilan zamonaviy frontend yaratish
- Node.js va Express bilan backend development
- MongoDB bilan ma'lumotlar bazasi boshqaruvi
- Authentication va authorization
- RESTful API yaratish va integration
- Production ga deploy qilish

Keyingi qadamlar:

1. O'z loyihalaringizni yarating
2. GitHub'da portfolio tuzing
3. Open source loyihalarga hissa qo'shing
4. Yangi texnologiyalarni o'rganishda davom eting
5. Developer community'da faol bo'ling

Muvaffaqiyatlar tilaymiz!