Study Guide / Cheat Sheet: Fullstack Web Development — Weeks 3 & 4

Full-Stack Integration, File Uploads, Testing, and Deployment

---

1. Frontend Auth Quick Reference

AuthContext Pattern

```
// src/context/AuthContext.jsx
import { createContext, useContext, useState, useEffect } from 'react';

const AuthContext = createContext(null);

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [token, setToken] = useState(null);
  const [isLoading, setIsLoading] = useState(true);

  // Check localStorage on mount
  useEffect(() => {
    const storedToken = localStorage.getItem('token');
    const storedUser = localStorage.getItem('user');
    if (storedToken && storedUser) {
      setToken(storedToken);
      setUser(JSON.parse(storedUser));
    }
    setIsLoading(false);
  }, []);

  const login = async (email, password) => {
    const res = await fetch('/api/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password }),
    });
    if (!res.ok) throw new Error('Login failed');
    const data = await res.json();
    localStorage.setItem('token', data.token);
    localStorage.setItem('user', JSON.stringify(data.user));
    setToken(data.token);
    setUser(data.user);
  };

  const logout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    setToken(null);
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, token, isLoading, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

// Custom hook
export const useAuth = () => useContext(AuthContext);
```

PrivateRoute Component

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

function PrivateRoute({ children }) {
  const { user, isLoading } = useAuth();

  if (isLoading) return <p>Loading...</p>;
  if (!user) return <Navigate to="/login" replace />;

  return children;
}

// Usage in routes:
<Route path="/dashboard" element={
  <PrivateRoute><DashboardPage /></PrivateRoute>
} />
```

Axios Auth Interceptor

```
// src/api/client.js
import axios from 'axios';

const api = axios.create({
  baseURL: '/api',
});

// Request interceptor: add Bearer token
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Response interceptor: handle 401 (auto-logout)
```

```
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('token');
      localStorage.removeItem('user');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);

export default api;
```

---

2. Full-Stack Integration Quick Reference

Vite Proxy Configuration

```
// vite.config.js
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/api': {
        target: 'http://localhost:3001',
        changeOrigin: true,
      },
    },
  },
});
```

CORS Configuration (Backend)

```
const cors = require('cors');

// Development: allow all
app.use(cors());

// Production: whitelist specific origins
app.use(cors({
  origin: ['http://localhost:5173', 'https://your-app.com'],
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization'],
}));
```

Loading/Error State Pattern

```
function TaskList() {
  const [tasks, setTasks] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchTasks = async () => {
      try {
        setIsLoading(true);
        setError(null);
        const { data } = await api.get('/tasks');
        setTasks(data);
      } catch (err) {
        setError(err.response?.data?.message || 'Failed to load tasks');
      } finally {
        setIsLoading(false);
      }
    };
    fetchTasks();
  }, []);

  if (isLoading) return <p>Loading tasks...</p>;
  if (error) return <p className="error">{error}</p>;
  if (tasks.length === 0) return <p>No tasks yet.</p>;

  return (
    <ul>
      {tasks.map(task => <TaskCard key={task.id} task={task} />)}
    </ul>
  );
}
```

Optimistic vs. Pessimistic Updates

Approach	How it works	Pros	Cons
Pessimistic	Update UI after server confirms	Data always in sync	Feels slower
Optimistic	Update UI immediately, roll back on error	Feels instant	Complex error handling

```
// Pessimistic (recommended for beginners)
const deleteTask = async (id) => {
  await api.delete(`/tasks/${id}`);     // Wait for server
  setTasks(tasks.filter(t => t.id !== id)); // Then update UI
};

// Optimistic
const deleteTask = async (id) => {
  const backup = [...tasks];
  setTasks(tasks.filter(t => t.id !== id)); // Update UI first
  try {
    await api.delete(`/tasks/${id}`);
  } catch {
    setTasks(backup); // Roll back on error
  }
};
```

---

3. File Uploads Quick Reference

Multer Setup (Backend)

```
const multer = require('multer');
const path = require('path');

// Configure disk storage
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    const uniqueName = Date.now() + '-' + file.originalname;
    cb(null, uniqueName);
  },
});

// File filter (images only)
const fileFilter = (req, file, cb) => {
  const allowed = ['image/jpeg', 'image/png', 'image/gif', 'image/webp'];
  if (allowed.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(new Error('Only image files are allowed'), false);
  }
};

const upload = multer({
  storage,
  fileFilter,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB max
});
```

Upload Route

```
// Single file upload
router.post('/users/avatar', auth, upload.single('avatar'), async (req, res) => {
  // req.file contains: { filename, path, mimetype, size }
  await prisma.user.update({
    where: { id: req.userId },
    data: { avatarUrl: `/static/${req.file.filename}` },
  });
  res.json({ avatarUrl: `/static/${req.file.filename}` });
});

// Serve uploaded files
app.use('/static', express.static('uploads'));
```

Frontend File Upload

```
function AvatarUpload() {
  const [preview, setPreview] = useState(null);

  const handleFileChange = (e) => {
    const file = e.target.files[0];
    if (file) {
      setPreview(URL.createObjectURL(file)); // Preview before upload
    }
  };

  const handleUpload = async (e) => {
    e.preventDefault();
    const file = e.target.querySelector('input[type="file"]').files[0];

    const formData = new FormData();
    formData.append('avatar', file); // 'avatar' must match multer field name

    const { data } = await api.post('/users/avatar', formData, {
      headers: { 'Content-Type': 'multipart/form-data' },
    });
    console.log('Uploaded:', data.avatarUrl);
  };

  return (
    <form onSubmit={handleUpload}>
      {preview && <img src={preview} alt="Preview" width="100" />}
      <input type="file" accept="image/*" onChange={handleFileChange} />
      <button type="submit">Upload</button>
    </form>
  );
}
```

---

4. Testing Quick Reference

Jest Basics

```
// describe groups related tests
describe('Math utils', () => {
  // it (or test) defines a single test
  it('should add two numbers', () => {
    expect(add(2, 3)).toBe(5);
  });

  it('should handle negative numbers', () => {
    expect(add(-1, 1)).toBe(0);
  });
});
```

Common Matchers

Matcher	Use Case	Example
`toBe(value)`	Exact equality (primitives)	`expect(1 + 1).toBe(2)`
`toEqual(obj)`	Deep equality (objects/arrays)	`expect({a: 1}).toEqual({a: 1})`
`toBeTruthy()`	Truthy check	`expect('hello').toBeTruthy()`
`toBeFalsy()`	Falsy check	`expect(null).toBeFalsy()`
`toContain(item)`	Array contains	`expect([1,2,3]).toContain(2)`
`toThrow()`	Function throws	`expect(() => fn()).toThrow()`
`toHaveLength(n)`	Array/string length	`expect([1,2]).toHaveLength(2)`
`toMatchObject(obj)`	Partial object match	`expect(res).toMatchObject({status: 'ok'})`

Setup and Teardown

```
describe('Task API', () => {
  let token;

  beforeAll(async () => {
    // Run once before all tests (e.g., register test user)
    const res = await request(app).post('/api/auth/register').send({
      username: 'testuser', email: 'test@test.com', password: 'password123'
    });
    token = res.body.token;
  });

  afterAll(async () => {
    // Run once after all tests (e.g., clean up database)
    await prisma.user.deleteMany({ where: { email: 'test@test.com' } });
    await prisma.$disconnect();
  });

  beforeEach(async () => {
    // Run before each test (e.g., reset state)
    await prisma.task.deleteMany({});
  });
});
```

Supertest Integration Tests

```javascript
const request = require('supertest');
const app = require('../app');

describe('POST /api/auth/register', () => {
  it('should register a new user', async () => {
    const res = await request(app)
      .post('/api/auth/register')
      .send({ username: 'newuser', email: 'new@test.com', password: 'pass123' });

    expect(res.status).toBe(201);
    expect(res.body).toHaveProperty('token');
    expect(res.body.user).toMatchObject({ username: 'newuser', email: 'new@test.com' });
  });

  it('should reject duplicate email', async () => {
    const res = await request(app)
      .post('/api/auth/register')
      .send({ username: 'dup', email: 'new@test.com', password: 'pass123' });

    expect(res.status).toBe(409);
  });

  it('should reject missing fields', async () => {
    const res = await request(app)
      .post('/api/auth/register')
      .send({ email: 'no-username@test.com' });

    expect(res.status).toBe(400);
  });
});

describe('GET /api/tasks', () => {
  it('should return tasks for authenticated user', async () => {
    const res = await request(app)
      .get('/api/tasks')
      .set('Authorization', `Bearer ${token}`); // Auth header

    expect(res.status).toBe(200);
    expect(Array.isArray(res.body)).toBe(true);
  });

  it('should return 401 without token', async () => {
    const res = await request(app).get('/api/tasks');
    expect(res.status).toBe(401);
  });
});
```

Test Database Isolation

```
# .env.test
DATABASE_URL=postgresql://user:pass@localhost:5432/webdev_test_db
JWT_SECRET=test_secret
```

```
// jest.config.js
module.exports = {
  testEnvironment: 'node',
  setupFiles: ['dotenv/config'], // or use dotenv in setup file
};
```

---

5. Deployment Quick Reference

Backend Dockerfile

```
FROM node:20-slim
WORKDIR /app
COPY package*.json ./
RUN npm ci --production
COPY . .
RUN npx prisma generate
EXPOSE 3001
CMD ["node", "src/app.js"]
```

Frontend Dockerfile (Multi-stage)

```
FROM node:20-slim AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
```

docker-compose.yml

```
version: '3.8'
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: taskflow
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports: ["5432:5432"]
    volumes: ["pgdata:/var/lib/postgresql/data"]

  backend:
    build: ./backend
    ports: ["3001:3001"]
    environment:
      DATABASE_URL: postgresql://postgres:postgres@db:5432/taskflow
      JWT_SECRET: your_secret_here
    depends_on: [db]

  frontend:
    build: ./frontend
    ports: ["80:80"]
    depends_on: [backend]

volumes:
  pgdata:
```

Environment Variables Checklist

Variable	Where	Example
`PORT`	Backend	`3001`
`DATABASE_URL`	Backend	`postgresql://user:pass@host:5432/db`
`JWT_SECRET`	Backend	Long random string
`JWT_EXPIRES_IN`	Backend	`24h`
`CORS_ORIGIN`	Backend	`http://localhost:5173`
`VITE_API_URL`	Frontend	`http://localhost:3001`

Production Checklist

- [ ] All environment variables set (no hardcoded secrets)
- [ ] `.env` in `.gitignore`
- [ ] `node_modules` in `.gitignore`
- [ ] `uploads/` in `.gitignore` (or handled by cloud storage)
- [ ] CORS configured for production origin
- [ ] JWT secret is strong and unique
- [ ] Database migrations applied
- [ ] Error handling returns safe messages (no stack traces)
- [ ] All tests passing
- [ ] Build completes without errors