Full-Stack Web Development Curriculum

Course Title: Full-Stack Web Application Development **Duration:** 4 Weeks — 3 sessions/week, 4 hours/session (48 contact hours)
Stack: React 18+, Node.js, Express.js, PostgreSQL, Prisma ORM **Capstone Project:** TaskFlow — A Project Management Application

---

Prerequisites

Students must have a solid foundation in:

- **HTML5 & CSS3:** Semantic HTML, Flexbox, Grid, responsive design
- **JavaScript (ES6+):** Variables, data types, functions, objects, arrays, promises, async/await, DOM manipulation
- **Git & GitHub:** clone, add, commit, push, pull, branching basics
- **Command Line:** Navigating directories, running commands in a terminal

---

Session Format (4 hours each)

Block	Duration	Description
Lecture & Demos	60 min	Core concepts, live coding examples, Q&A
Guided Lab	60 min	Instructor-led hands-on exercise
Independent Lab + TaskFlow	60 min	Apply concepts to standalone project + TaskFlow milestone
TaskFlow Sprint + Wrap-up	60 min	Dedicated capstone work, pair programming, review

---

Grading Breakdown

Component	Weight	Details
Lab Exercises	25%	In-session labs (best 10 of 11)
Take-Home Assignments	25%	Applied exercises extending session topics
Midpoint Practical (Session 6)	15%	Build a feature from spec under time constraints
Capstone Project: TaskFlow	25%	Final application (code + functionality + quality)
Capstone Presentation	10%	Live demo + technical Q&A (Session 12)

Course Objectives

1. Build interactive user interfaces with React 18+ (components, hooks, routing)
2. Implement server-side logic with Node.js and Express.js
3. Design RESTful APIs with proper HTTP methods and status codes
4. Apply middleware patterns for logging, validation, auth, and error handling
5. Design relational database schemas with PostgreSQL
6. Use Prisma ORM for type-safe database access and migrations
7. Implement authentication with bcrypt password hashing and JWT tokens
8. Integrate frontend and backend with proper CORS, error handling, and auth flows
9. Handle file uploads and static file serving

10. Write unit and integration tests with Jest and Supertest

Capstone Project: TaskFlow

TaskFlow is a project management application built incrementally across all 12 sessions:

- User registration and login (JWT authentication)
- Task CRUD with status tracking (pending, in-progress, completed)
- User profile with avatar upload
- Responsive React frontend consuming a RESTful API
- Prisma ORM with PostgreSQL persistence
- Protected routes (frontend and backend)
- Test coverage for critical paths

Weekly Breakdown

Week 1: Frontend Foundations

Session 1: React Foundations — Components, Props, State

Objectives: 1

Lecture Topics:

- Course overview and full-stack landscape
- Setting up Node.js, npm, and Vite
- What is React? Virtual DOM, component-based architecture
- JSX syntax and functional components
- Props for data flow between components
- `useState` hook for managing component state

Lab Exercise (2 hours):

- Initialize a Vite React project
- Build `CounterDisplay` component with `startValue` prop and `useState`
- Add increment/decrement buttons with `onClick` handlers
- Create multiple counter instances to demonstrate reusability
- **Expected Output:** Page with two independent counters that track state separately

TaskFlow Milestone: Set up the TaskFlow React project with Vite. Create `Header`, `Footer`, and a static `TaskCard` component that displays task title and status via props.

Code: `week-1/session-1-react-foundations/`

Session 2: State Management — useEffect, Forms, Routing

Objectives: 1

Lecture Topics:

- Handling user events (onClick, onChange, onSubmit)
- Controlled form components
- `useEffect` hook — side effects, dependency arrays, cleanup

- Conditional rendering patterns (loading, empty, data states)
- Lists and keys
- React Router basics (`react-router-dom` v6)

Lab Exercise (2 hours):

- Build a todo list with simulated async data fetching via `useEffect` + `setTimeout`
- Add conditional rendering: loading spinner, empty state, todo list
- Create `AddTodoForm` with controlled input and validation
- Implement mark-complete toggle with strikethrough styling
- **Expected Output:** Loading state transitions to todo list; form adds items; toggle completes them

TaskFlow Milestone: Create `TaskInput` form and `TaskList` component. Use `useState` for a task array. Implement add/display/toggle-complete with mock data. Add React Router with Home and Tasks pages.

Code: `week-1/session-2-state-and-forms/`

Session 3: Node.js + Express Foundations

Objectives: 2, 3, 4

Lecture Topics:

- Node.js event-driven architecture and non-blocking I/O
- npm project initialization and dependency management
- Express.js server setup and route definitions (GET, POST, PUT, DELETE)
- Request parameters: `req.params` , `req.query` , `req.body`
- Middleware pattern: `express.json()` , `cors` , custom logging, error handling
- HTTP status codes and JSON responses

Lab Exercise (2 hours):

- Create an Express server with REST endpoints for an in-memory items API
- Add custom logging middleware, input validation, 404 handler, error handler
- Test all endpoints with curl or Postman
- **Expected Output:** Full CRUD API with middleware chain, proper status codes, error responses

TaskFlow Milestone: Create the TaskFlow backend with Express. Implement `GET /api/tasks` , `GET /api/tasks/:id` , `POST /api/tasks` , `PUT /api/tasks/:id` , `DELETE /api/tasks/:id` using in-memory storage. Add cors, logging, validation, and error handling middleware.

Code: `week-1/session-3-express-foundations/`

Week 2: Database + Backend Architecture

Session 4: PostgreSQL Schema Design + Prisma Setup

Objectives: 5, 6

Lecture Topics:

- Introduction to relational databases and PostgreSQL
- SQL fundamentals: CREATE TABLE, INSERT, SELECT, UPDATE, DELETE
- Data types, primary keys, foreign keys, constraints
- Table relationships (one-to-many)
- Prisma architecture: schema file, client, migrations

- `prisma/schema.prisma` syntax: models, fields, relations

Lab Exercise (2 hours):

- Install PostgreSQL and create a database
- Write SQL to create `users` and `tasks` tables with foreign key relationship
- Insert sample data and practice SELECT, WHERE, JOIN, ORDER BY
- Initialize Prisma, define models, run first migration
- **Expected Output:** Populated database with working SQL queries; Prisma schema matching SQL tables

TaskFlow Milestone: Design the TaskFlow database schema. Create `users` and `tasks` tables. Initialize Prisma with matching models. Run migration. Provide `.sql` seed scripts.

Code: `week-2/session-4-postgresql-and-prisma/`

Session 5: Prisma CRUD + Backend Refactor

Objectives: 2, 6

Lecture Topics:

- Prisma Client CRUD: `create`, `findMany`, `findUnique`, `update`, `delete`
- Prisma relations: `include`, nested writes
- Refactoring from in-memory to database-backed routes
- Environment variables with `dotenv`
- Express Router for modular route organization
- Database error handling patterns

Lab Exercise (2 hours):

- Create `src/prisma.js` with PrismaClient singleton
- Refactor all CRUD routes to use Prisma Client with proper error handling
- Organize routes into separate Express Router modules
- Test all endpoints — verify data persists across server restarts
- **Expected Output:** All CRUD operations use Prisma; data persists in PostgreSQL

TaskFlow Milestone: Refactor TaskFlow backend from in-memory to Prisma. Split routes into `routes/tasks.js` and `routes/users.js`. All task CRUD uses Prisma Client. Data persists across restarts.

Code: `week-2/session-5-prisma-crud/`

Session 6: Authentication — bcrypt + JWT

Objectives: 7

Lecture Topics:

- Authentication vs. authorization
- Password hashing with bcrypt (salting, cost factor)
- JSON Web Tokens: structure, signing, verification
- Registration flow: validate → hash → store → respond
- Login flow: find user → compare hash → generate JWT
- Auth middleware: extract token → verify → attach user to request
- Token expiration and security best practices

Lab Exercise (2 hours):

- Implement `POST /api/auth/register` with bcrypt password hashing
- Implement `POST /api/auth/login` with bcrypt.compare and JWT generation
- Create auth middleware that verifies Bearer tokens
- Protect task routes — only authenticated users can access
- **Expected Output:** Register creates hashed user; login returns JWT; protected routes require valid token

Midpoint Practical (graded): Build a small authenticated API feature from a provided spec within time constraints.

TaskFlow Milestone: Add register and login endpoints. Hash passwords with bcrypt, issue JWTs on login. Create auth middleware. Protect all `/api/tasks` routes. Scope tasks to authenticated user (`WHERE user_id = req.userId`).

Code: `week-2/session-6-authentication/`

Week 3: Full-Stack Integration

Session 7: Frontend Authentication State

Objectives: 1, 7

Lecture Topics:

- Client-side token storage: localStorage vs. cookies vs. sessionStorage
- Security considerations (XSS, CSRF)
- React Router protected routes
- Creating AuthContext with Context API
- Login/logout UI flows
- Persisting auth state across page refreshes

Lab Exercise (2 hours):

- Create `AuthContext` with user state, login/logout functions
- Build login and registration page components
- Implement `PrivateRoute` component that redirects unauthenticated users
- Store JWT in localStorage, restore on mount
- **Expected Output:** Login stores token and redirects; refresh preserves session; logout clears state

TaskFlow Milestone: Build TaskFlow login and registration pages. Create AuthContext for global auth state. Store JWT in localStorage. Implement protected routes. Add logout to Header.

Code: `week-3/session-7-frontend-auth/`

Session 8: Full-Stack Integration

Objectives: 1, 3, 8

Lecture Topics:

- Making API calls from React with axios
- Creating an API client with auth interceptor
- CORS configuration: origin whitelist
- Loading states and error handling patterns
- Handling 401 responses (auto-logout)
- Optimistic vs. pessimistic UI updates

Lab Exercise (2 hours):

- Create `src/api/client.js` with axios instance and Bearer token interceptor
- Wire TaskList to fetch from API with loading and error states
- Wire TaskInput to POST new tasks
- Implement edit and delete through the UI
- Handle 401 in response interceptor (auto-logout)
- **Expected Output:** Full end-to-end flow: register → login → CRUD tasks through UI

TaskFlow Milestone: Fully integrate TaskFlow frontend and backend. Users can register, login, create/view/update/delete their own tasks through the UI. Loading indicators on all API calls. Error messages for failures. 401 auto-logout.

Code: `week-3/session-8-fullstack-integration/`

Session 9: File Uploads + Feature Sprint

Objectives: 9

Lecture Topics:

- File upload concepts: multipart/form-data
- `multer` middleware: disk storage, file filtering, size limits
- Serving static files with `express.static()`
- Security: file type validation, path traversal prevention
- Storing file metadata in the database

Lab Exercise (2 hours):

- Configure multer with disk storage and file type filtering
- Create `POST /api/users/:id/avatar` upload endpoint
- Serve uploaded files via `express.static()`
- Build React file upload component with preview
- **Expected Output:** Avatar uploads to server, URL stored in DB, displayed in UI

TaskFlow Milestone: Add profile picture upload to TaskFlow. Create upload endpoint with multer. Store filename in user record via Prisma. Serve uploads statically. Display avatar in user profile UI.

Code: `week-3/session-9-file-uploads/`

Week 4: Testing, Deployment, and Capstone

Session 10: Testing

Objectives: 10

Lecture Topics:

- Why test? Unit vs. integration vs. end-to-end
- Jest fundamentals: describe, it, expect, matchers
- Testing Express APIs with Supertest
- Test database isolation (separate `.env.test`)
- Setup/teardown: `beforeAll`, `afterAll`, `beforeEach`

Lab Exercise (2 hours):

- Write unit tests for utility/validation functions
- Write integration tests for auth endpoints (register, login)
- Write integration tests for task CRUD endpoints (with auth)

- Configure test database isolation
- **Expected Output:** All tests pass with isolated test database

TaskFlow Milestone: Write tests for TaskFlow: unit tests for validators, integration tests for register, login, and all task CRUD endpoints. Configure test database.

Code: `week-4/session-10-testing/`

Session 11: Deployment Preparation + Polish

Objectives: All

Lecture Topics:

- Environment variables for production
- Build scripts and production configuration
- Deployment platforms: Render, Railway, Vercel
- Frontend build and static hosting
- Database hosting (Supabase, Railway, Render)
- Final polish: validation hardening, UX improvements

Lab Exercise (2 hours):

- Configure production environment variables
- Build React frontend for production
- Fix top bugs and polish UX
- Prepare deployment configuration
- **Expected Output:** Production-ready application with deploy config

TaskFlow Milestone: Finalize TaskFlow: fix bugs, polish UI, add final validations. Create production build scripts. Write README with setup instructions and API documentation.

Code: `week-4/session-11-deployment-and-polish/`

Session 12: Capstone Presentations

Objectives: All (1-10)

Presentation Format (20 min each):

- Live demo of TaskFlow application
- Code walkthrough of one interesting feature
- Discussion of challenges faced and solutions
- Q&A from instructor and peers

Capstone Grading Rubric (250 points):

Criteria	Points
User registration and login (bcrypt + JWT)	35
Task CRUD — full functionality	35
Database schema design and Prisma integration	30
RESTful API design (correct methods, status codes)	25
Middleware usage (auth, validation, error handling)	25

Frontend UI (React components, state management)	25
Frontend-backend integration (auth headers, CORS)	25
File upload functionality	20
Testing (unit + integration)	15
Code quality and organization	15

Technology Reference

Technology	Version	Purpose
React	18+	Frontend UI framework
Vite	5+	Frontend build tool
React Router	6+	Client-side routing
Node.js	20+ LTS	JavaScript runtime
Express.js	4.x	Backend web framework
PostgreSQL	15+	Relational database
Prisma	5.x	ORM (database toolkit)
bcrypt	5.x	Password hashing
jsonwebtoken	9.x	JWT authentication
multer	1.x	File upload handling
axios	1.x	HTTP client for frontend
cors	2.x	Cross-origin resource sharing
dotenv	16.x	Environment variable management
Jest	29.x	Testing framework
Supertest	6.x	HTTP assertion library

Recommended Resources

- **MDN Web Docs** — JavaScript, HTTP, Web APIs reference
- **React Official Docs** — react.dev
- **Express.js Guide** — expressjs.com
- **Prisma Docs** — prisma.io/docs
- **PostgreSQL Tutorial** — postgresqltutorial.com

Academic Integrity

All code submitted must be the student's own work. Use of AI assistants (ChatGPT, Claude, Copilot) is permitted for learning and debugging but not for generating entire assignments. Students must be able to explain every line of code they submit.