MENEDŻERSKA AKADEMIA NAUK STOSOWANYCH W WARSZAWIE

The use of web system in information systems

Meeting 4 – Lab

19.12.2025 15:15 – 19:45

Meeting agenda

- Research, Low-Fidelity Prototyping, and First Testing
- Create Low-Fidelity Wireframes
- Peer Review and First Usability Test
- Iterative Improvement of Wireframes
- Project Setup, Component Architecture, and Initial Implementation

Meeting 4 - Objective

Build and deploy the **server-side backend** for your web-based information system and connect it with the existing front-end. You will expose RESTful API endpoints to handle CRUD operations, implement basic authentication, and exchange real data between client and server. This marks the transition from a "static demo" into a working information system prototype.

Part I - Define API Contract and Data Models

- Based on your previous data model (ERD), identify at least two resources to expose via REST API (e.g., /users, /records, /bookings).
- For each resource, define:
  - Available endpoints (e.g., GET /records, POST /records)
  - Expected JSON structure for requests/responses
  - HTTP status codes and error handling plan
- Create OpenAPI/Swagger specification or document the API in Postman.

Helpers:

- https://swagger.io/specification/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods

Deliverable:

- API contract or OpenAPI schema + JSON sample files

Part II - Backend Setup and Framework Selection

- Choose a backend stack (example: **Node.js + Express** or **Python + Flask/FastAPI**)
- Set up:
  - REST framework
  - Middleware (CORS, body parsers, JSON)
  - Routing and endpoint structure
  - Version control (Git + GitHub)
- Write first working endpoint returning mock data.

Helpers:

- https://fastapi.tiangolo.com/tutorial/first-steps/
- https://expressjs.com/en/starter/basic-routing.html

Deliverable:

- Running server with one endpoint

Part III - Implement Core Logic

- Implement:
  - At least two full CRUD endpoints for your main resource
  - Input validation (e.g., using Pydantic, Joi, or schema checks)
  - In-memory or SQLite database for testing
  - Meaningful error responses (400, 404, 500)
- Create a simple Postman test collection or curl test scripts

Deliverable:

- Backend running with working API, sample test results (e.g., Postman screenshots)

Part IV - Prepare API for Front-End Integration

- Enable CORS
- Define clear URL structure and CORS policy
- Deploy (optional) using Render, Railway, or local tunnel (ngrok)
- Update your frontend to prepare for fetch/axios integration

Deliverable:

- Working local API + documentation of endpoint behavior

Break

Part V - Frontend–Backend Integration

- Replace mock data with real API calls using fetch or axios
- Connect forms (e.g., login, create/edit forms) with backend
- Show real-time updates on UI (e.g., after adding/editing/deleting data)
- Handle frontend error states (e.g., validation failed, not found)

Deliverable:

- Fully connected frontend, screen recording or screenshot proof

Part VI - Final Testing and Deployment

Test all user scenarios end-to-end:

- Add new record via frontend → show in table
- Edit or delete record
- Handle auth errors or bad inputs

Deliverable:

- Test plan + screenshots

Grading criteria

- API Contract and Resource Design - Completeness and correctness of API specification (OpenAPI/Postman), clear endpoint structure, realistic sample data, proper HTTP method use
- Backend Setup and Architecture - Proper initialization of backend project (Express/FastAPI/etc.), file/folder structure, initial route setup, middleware configuration
- Core API Functionality (CRUD & Validation) - At least two fully implemented resources, working CRUD endpoints, validation of inputs, proper use of status codes (200, 400, 404, 500)
- Front-End Integration - Connection of forms and UI to backend via HTTP calls, real-time data flow in UI (e.g. reload list after POST), clear error/success handling