Advance Web Technologies Lab Assignments

Introduction

Based on the syllabus, the four assignments should **not be four unrelated mini-projects**. They should be four stages of **one progressively complex real-world problem**, with each assignment introducing only the technologies that students have already learned.

The syllabus explicitly progresses from **Node.js → FS → MongoDB → Mongoose → Express → REST API → authentication/JWT → Angular/MEAN → React/MERN**.

The CLO mapping also strongly supports this progression: **CLO-2 covers Units 2–4, CLO-3 covers REST API, and CLO-4 covers MEAN/MERN**, while the lab CLOs focus on developing REST servers and dynamic clients.

Proposed Four-Assignment Structure

Overall Real-World Case Study

Smart Campus Service Management System

A university receives hundreds of requests every day from students and staff, such as:

- Classroom/equipment problems
- IT support requests
- Maintenance complaints
- Transport requests
- Library issues
- Lost & found reports
- Facility/service requests

The system will gradually evolve from a **simple Node.js server** into a **complete MEAN/MERN-style web application**.

This domain is useful because it is realistic enough to involve **data modelling, APIs, persistence, authentication, business rules, and dynamic clients**, but the individual assignments can remain small.

Assignment 1 — Node.js-Based Service Request Server

Title

Assignment 1: Build a Basic Service Request Server Using Node.js

Syllabus Position

Weeks/Lectures 3–6

Relevant topics include:

- Node.js architecture and modules
- Event loop, callbacks and events
- NPM
- HTTP Server
- URL module
- File System
- Files and Streams

Problem

The university currently receives service complaints through different channels. Develop a basic server that allows a user to submit and retrieve service requests.

Each request should contain:

```
Request ID
Student/Staff Name
Category
Description
Priority
Date
Status
```

Requirements

Students must:

1. Create an HTTP server using Node.js.

2. Implement basic URL-based routing.
3. Support:
    - GET /requests
    - GET /requests/:id
    - POST /requests
4. Store requests in a local JSON file.
5. Read requests using Node.js File System APIs.
6. Add new requests to the file.
7. Return responses in JSON format.
8. Handle invalid URLs and invalid request IDs.

Constraints

Do NOT use:

- Express
- MongoDB
- Mongoose
- Angular
- React

The purpose is to make students understand what Express and database technologies will later solve.

Expected Output

A Node.js application demonstrating:

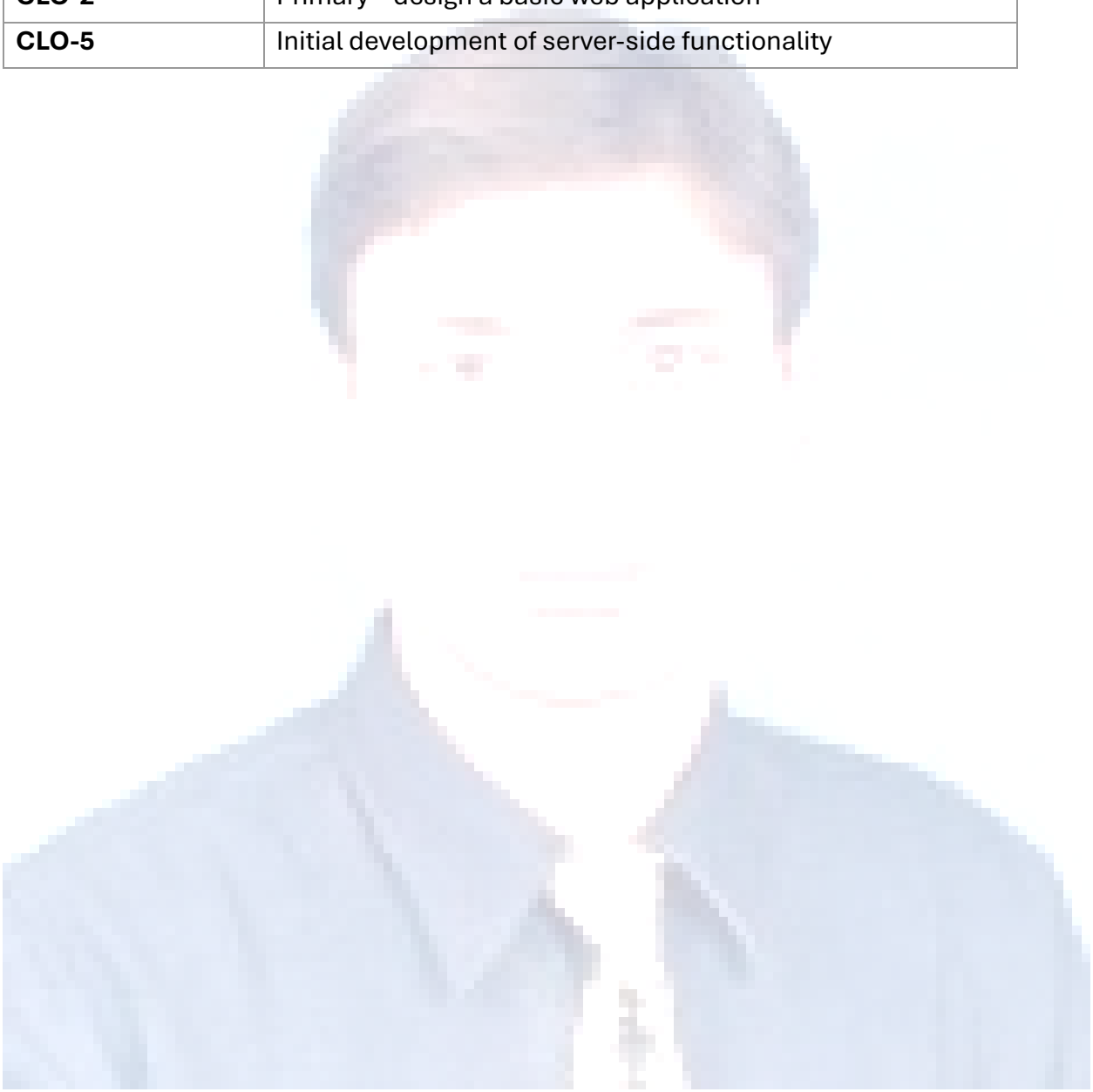
```
Client
↓
HTTP Request
↓
Node.js HTTP Server
↓
URL Processing
↓
File System
↓
JSON Response
```

Estimated Workload

One week maximum

CLO Alignment

CLO	Alignment
CLO-1	Understanding Node.js concepts
CLO-2	Primary – design a basic web application
CLO-5	Initial development of server-side functionality



Assignment 2 — Persistent Service Management Using MongoDB & Mongoose

Title

Assignment 2: Convert the Service Request Server into a Database-Driven Application

Syllabus Position

Weeks/Lectures 7–12

Students have now learned:

- MongoDB
- Collections
- Schemas
- Aggregation
- MongoDB Node.js Client
- Mongoose
- CRUD
- Express
- Middleware
- Routing
- Routers

Problem

The university has outgrown the JSON-file solution. Thousands of service requests must now be stored and managed efficiently.

Students must convert Assignment 1 into a **database-driven application**.

Requirements

Create MongoDB collections for:

Users
ServiceRequests
Categories

Implement CRUD operations for service requests:

- Create request
- Retrieve all requests
- Retrieve request by ID
- Update request
- Delete request

Mongoose Requirements

Students must:

1. Create a Mongoose schema for ServiceRequest.
2. Create a Mongoose model.
3. Connect Node.js to MongoDB.
4. Implement CRUD operations.
5. Implement at least **one aggregation query**, for example:

Find the number of unresolved requests in each category.

Example:

```
IT Support    24
Maintenance   18
Library       7
Transport     11
```

Express Introduction

Students should now replace the manually created HTTP routing from Assignment 1 with Express:

```
Express
 ├── Router
 ├── Middleware
 └── Controller logic
        ↓
    Mongoose
        ↓
    MongoDB
```

Expected Output

A working Express + MongoDB application capable of managing service requests.

Estimated Workload

One week maximum

CLO Alignment

CLO	Alignment
CLO-2	**Primary** – design application using advanced technologies
CLO-3	Preparation for REST API design
CLO-5	**Primary** – server development

Assignment 3 — REST API for Smart Campus Services

Title

Assignment 3: Design and Implement a Secure REST API

Syllabus Position

Weeks/Lectures 13–20

This is where the syllabus explicitly introduces:

- REST principles
- Statelessness
- Idempotency
- HTTP method selection
- Resources and URIs
- REST API design
- REST implementation
- CORS
- XSS
- DDoS overview
- Authentication/Authorization
- Sessions
- Passport
- JWT

Problem

The university wants multiple applications to use the same service-request system:

```
Web Application
Mobile Application
Admin Dashboard
Future Applications
        ↓
     REST API
        ↓
     Server
```

Students must transform Assignment 2 into a **RESTful API**.

Required API

Students should design and implement endpoints such as:

Method	Endpoint	Purpose
GET	/api/requests	Get requests
GET	/api/requests/:id	Get one request
POST	/api/requests	Create request
PUT	/api/requests/:id	Update request
DELETE	/api/requests/:id	Delete request

Business Requirement

Introduce a simple workflow:

```
Submitted
    ↓
Assigned
    ↓
In Progress
    ↓
Resolved
    ↓
Closed
```

Students should implement basic validation so that inappropriate status transitions are rejected.

Authentication

Implement **JWT-based authentication** for at least two roles:

```
Student
Service Administrator
```

Example:

```
Student
  ↓
Can create/view own requests

Administrator
  ↓
Can view/update/assign requests
```

API Security

Implement basic:

- CORS configuration
- Input validation
- Authentication middleware
- Authorization middleware

Expected Output

A documented REST API that can be tested using Postman/Thunder Client.

Estimated Workload

One week maximum

CLO Alignment

This assignment has the strongest direct mapping to:

CLO	Alignment
CLO-3	**Primary** – design REST API based web server
CLO-5	**Primary** – develop REST API server
CLO-2	Supporting

This directly matches the syllabus CLO for REST API development.

Assignment 4 — Dynamic Smart Campus Client

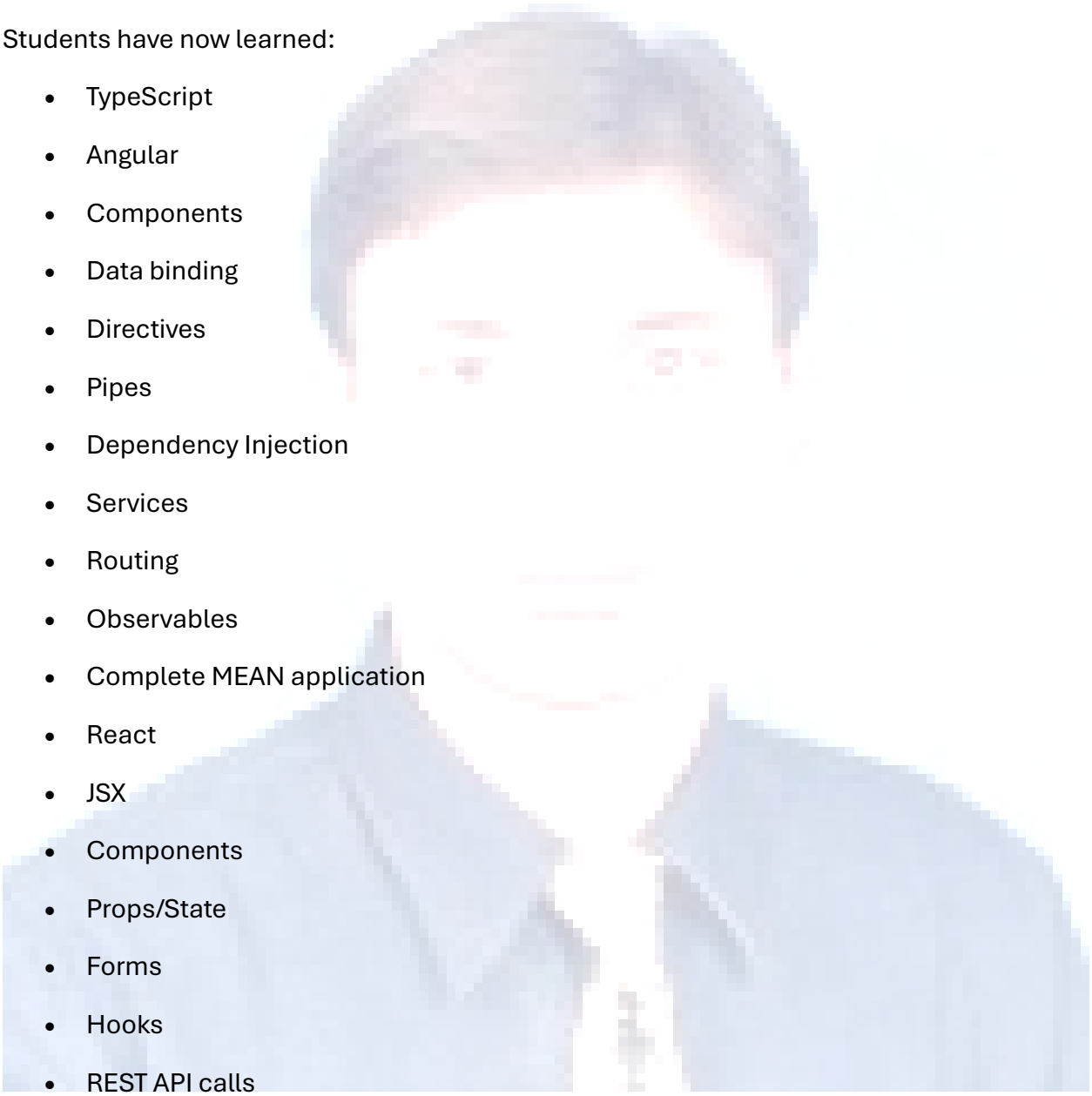
Title

Assignment 4: Develop a Dynamic Client for the Smart Campus REST API

Syllabus Position

Weeks/Lectures 21–32

Students have now learned:



- TypeScript
- Angular
- Components
- Data binding
- Directives
- Pipes
- Dependency Injection
- Services
- Routing
- Observables
- Complete MEAN application
- React
- JSX
- Components
- Props/State
- Forms
- Hooks
- REST API calls
- Redux/Thunk

- MERN application development

Problem

The REST API from Assignment 3 is now available to university users. Develop a **dynamic web client** that consumes the API.

Recommended Approach

Use **Angular**, because it naturally follows the MEAN-stack portion of the syllabus.

The client should provide:

Student Dashboard

```
--------------------------------
 Smart Campus Services
--------------------------------

My Requests

REQ-101   IT Support      Open
REQ-102   Maintenance     Progress
REQ-103   Library         Closed

[ + New Request ]
```

Students should be able to:

- Login
- View their requests
- Submit a request
- View request details
- Track request status

Administrator Dashboard

Administrators should be able to:

- View all requests
- Filter by category
- Filter by status

- Update request status
- Assign requests
- View basic statistics

Technical Requirements

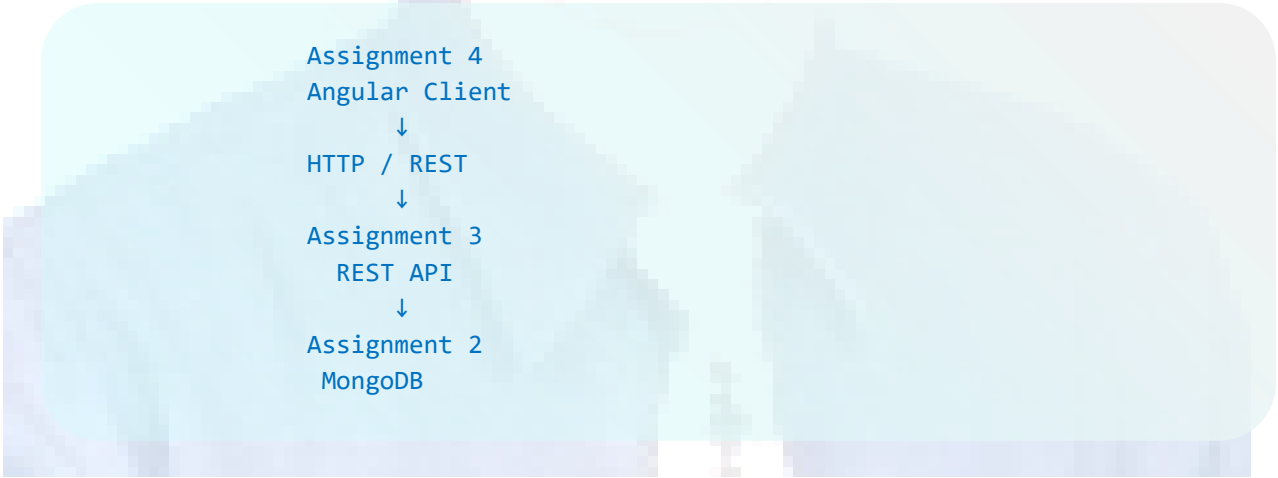
Students should demonstrate:

- Angular components
- TypeScript
- Angular services
- Routing
- Data binding
- Forms
- HTTP communication
- Observable-based API calls
- Dependency Injection

Important Restriction

Students **do not need to rebuild the backend**.

They reuse Assignment 3:



Estimated Workload

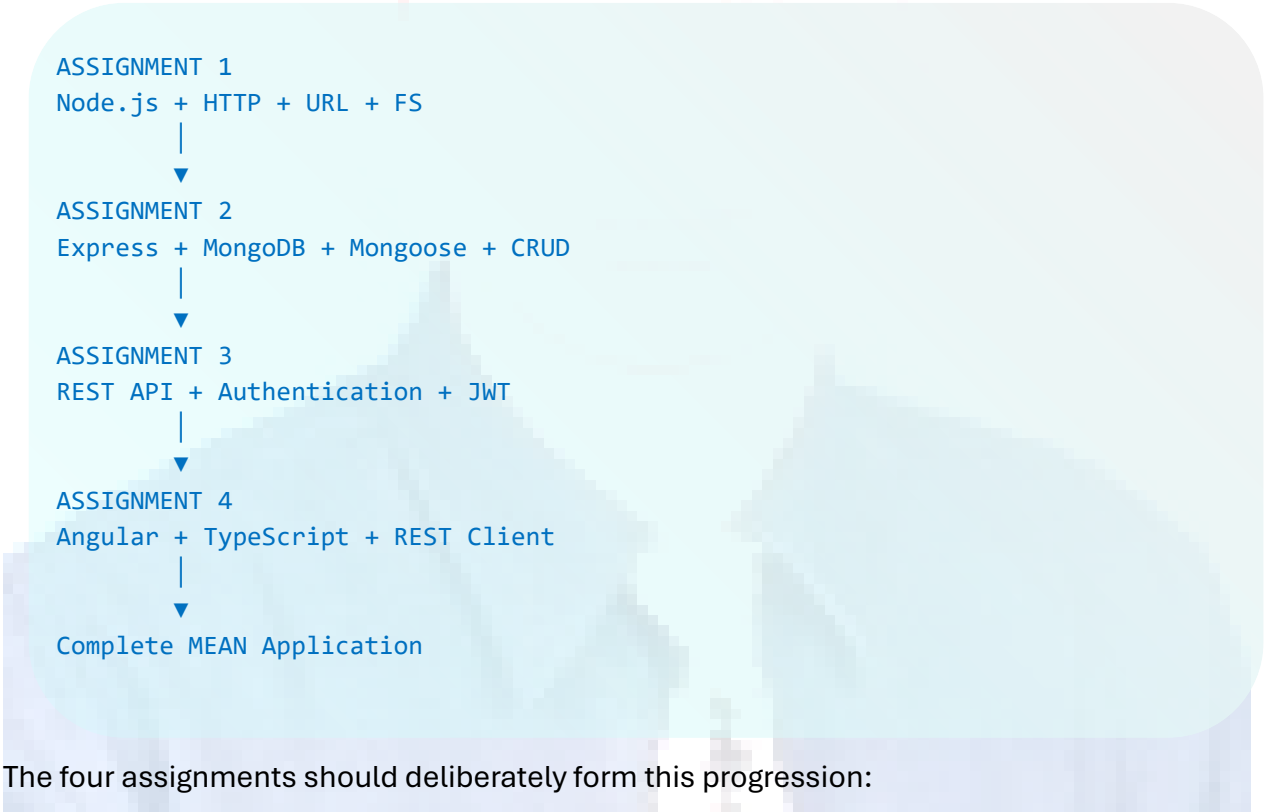
One week maximum

CLO Alignment

CLO	Alignment
CLO-4	**Primary** – MEAN/MERN technologies
CLO-6	**Primary** – develop dynamic clients
CLO-3	Supporting – consume REST API

This directly corresponds to the lab CLO requiring students to **develop dynamic clients using modern development technologies**.

The Progression Is the Important Part



The four assignments should deliberately form this progression:

This is much better pedagogically than giving students four independent problems.

Assignment-to-Syllabus Mapping

Assignment	Main Syllabus Topics	Primary CLO	Max Workload
1. Basic Service Server	Node.js, HTTP, URL, FS, JSON	CLO-2 / CLO-5	3 hrs
2. Database Service System	MongoDB, Mongoose, CRUD, Express	CLO-2 / CLO-5	3 hrs
3. REST Service API	REST, Express, Security, JWT	**CLO-3 / CLO-5**	3 hrs
4. Dynamic Web Client	TypeScript, Angular, Services, Routing, Observables, MEAN	**CLO-4 / CLO-6**	3 hrs

The progression also respects the syllabus sequence rather than asking students to use technologies before they have been taught. The syllabus places Node.js and FS before MongoDB/Mongoose, Express before REST, and Angular/MEAN after the REST and authentication material.

Recommended Assessment Design

Since the syllabus assigns **25 marks to lab assignments**, its appropriate to divide the four assignments equally:

Assignment	Marks
Assignment 1	6
Assignment 2	6
Assignment 3	7
Assignment 4	6
Total	**25**

This also fits the syllabus assessment structure, which allocates **25 marks to Lab**

Assignments.

One important design decision

I would **not make Assignment 4 a complete MERN application**. The syllabus teaches both Angular/MEAN and React/MERN, but requiring students to implement a full MERN application within a 3-hour assignment would make the workload unrealistic. Instead, Assignment 4 should establish the **dynamic client using Angular/MEAN**, while React/MERN can be reinforced through the lab project or final practical work. The syllabus itself reserves a **Lab Project** separately from the assignments.

This gives the four assignments a clean educational narrative:

"Build it → persist it → expose it → consume it."

That is, in my view, the strongest structure for this syllabus while keeping **each assignment genuinely achievable within one week**.