Lab Assignment #3 – Micro-Frontends and Microservices with GraphQL

Due Date: Week 10, Wednesday, 10:30am.

Purpose: The purpose of this assignment is to:

- Extend the existing **micro-frontends and microservices architecture to develop a community engagement system** (Software Engineering Technology students) or **a game progress tracking system** (Game Programming students).
- Utilize GraphQL for communication between micro-frontends and microservices.
- **Align the implementation with the ongoing group project**, allowing students to optimize their workload and build upon their existing work.
- Encourage collaboration and integration of components across the project to streamline development and enhance learning.

References: Read the reference textbooks, lecture slides, and class examples. This material provides the necessary information that you need to complete the exercises.

Be sure to read the following general instructions carefully:

- This assignment may be completed using the **pair programming technique** (https://en.wikipedia.org/wiki/Pair_programming).
- See the naming and **submission rules** at the end of this document
- You will have to **provide a demonstration for your solution** and upload the solution to Luminate course shell.

Exercise 1: For Software Engineering Technology students

In this exercise you will set up the project infrastructure and begin backend development, using cutting-edge technologies, including Micro Frontends with Vite Module Federation for the React user interface and Microservices with Express.js, Apollo Server, and Graph QL for the backend.

I. Backend Development

1. **User Authentication Microservice -** Develop a microservice using Express, Apollo Server, and GraphQL to handle **user registration and login**.
    a) Implement MongoDB schema for users.
    b) Implement mutations for **signup**, **login**, and **logout**.
    c) Ensure security measures for user authentication (e.g., hashing passwords).
2. **Community Engagement Microservice** - Develop a microservice using Express, Apollo Server, and GraphQL to handle news, discussions, and help requests.
    a) Implement MongoDB schema for community posts and help requests.
    b) Implement GraphQL queries and mutations for community interactions.

Definitions of fields

User Schema Fields

Field Name	Type	Required	Unique	Default	Description
username	`String`	Yes	Yes	No	Unique username for each user.
email	`String`	Yes	Yes	No	User email address.
password	`String`	Yes	No	No	User password stored securely (hashed).
role	`String`	Yes	No	No	Defines user permissions. Allowed values: `'resident'`, `'business_owner'`, `'community_organizer'`.
createdAt	`Date`	No	No	`Date.now`	Timestamp for when the user was created.

Community Post Schema

Field Name	Type	Required	Unique	Default	Description
author	`ObjectId` (ref: `User`)	Yes	No	No	Reference to the user who created the post.
title	`String`	Yes	No	No	Title of the post.
content	`String`	Yes	No	No	Main body of the post.
category	`String`	Yes	No	No	Defines the type of post. Allowed values: `'news'`, `'discussion'`.
aiSummary	`String`	No	No	No	AI-generated summary of long discussions.
createdAt	`Date`	No	No	`Date.now`	Timestamp of post creation.
updatedAt	`Date`	No	No	No	Timestamp of the last update.

Help Request Schema Fields

Field Name	Type	Required	Unique	Default	Description
author	`ObjectId` (ref: `User`)	Yes	No	No	Reference to the user who created the help request.
description	`String`	Yes	No	No	Description of the help request.
location	`String`	No	No	No	Used for location-based help requests.
isResolved	`Boolean`	No	No	`false`	Indicates whether the help request has been resolved.
volunteers	`Array<ObjectId>` (ref: `User`)	No	No	No	Users who volunteered for the request.
createdAt	`Date`	No	No	`Date.now`	Timestamp of request creation.
updatedAt	`Date`	No	No	No	Timestamp of the last update.

II. Initial Frontend Development

1. **Authentication Micro Frontend -** Develop a micro frontend using React Vite and Apollo Client.
   a) Create a micro frontend responsible for user authentication (signup, login, logout).
   b) Integrate with the Authentication Microservice for user-related operations.

2. **Community Engagement Micro Frontend -** Develop a micro frontend using React Vite and Apollo Client.
   a) Create a micro frontend responsible for handling discussions, news, and help requests.

b) Integrate with the Community Engagement Microservice.

Use **functional components**, client-side composition, and **React Hooks** for the Micro Frontends. Design a visually appealing and user-friendly UI.

(10 marks)

Evaluation:

Functionality(including code explanation during demonstration):	
Micro frontends (Authentication, Community Engagement)	30%
MongoDB database (config files, models)	5%
GraphQL Microservices (Authentication, Community Engagement)	30%
Integration using Vite Module Federation plugin	20%
Friendliness (using CSS to align the React elements, React-Bootstrap, etc.)	5%
Code **demonstration** and brief explanation during demonstration in class	10%
Total	**100%**

Exercise 2: For Game – Programming students

In this exercise you will set up the project infrastructure and begin backend development, using cutting-edge technologies, including Micro Frontends with Vite Module Federation for the React user interface and Microservices with Express.js, Apollo Server, and Graph QL for the backend.

I. Backend Development

1. **User Authentication Microservice -** Develop a microservice using Express, Apollo Server, and GraphQL to handle **user registration and login**.
   a. Implement MongoDB schema for users.
   b. Implement mutations for **signup**, **login**, and **logout**.
   c. Ensure security measures for user authentication (e.g., hashing passwords).

2. **Game Progress Microservice** - Develop a microservice using Express, Apollo Server, and GraphQL for **tracking player achievements and leaderboards**.
   a. Implement MongoDB schema for game progress.
   b. Implement GraphQL queries and mutations

Definition of fields

User Schema Fields

Field Name	Type	Required	Unique	Default	Description
`username`	`String`	Yes	Yes	No	Unique username for each user.
`email`	`String`	Yes	Yes	No	User email address.
`password`	`String`	Yes	No	No	User password stored securely (hashed).
`role`	`String`	Yes	No	`"player"`	Defines user permissions. Allowed values: `"player"`, `"admin"`.
`createdAt`	`Date`	No	No	`Date.now`	Timestamp for when the user was created.

Game Progress Schema Fields

Field Name	Type	Required	Default	Description
`_id`	ObjectId	Yes (Auto)	Auto	Unique identifier for each document.
`userId`	ObjectId (ref: User)	Yes	No	References the User model to link game progress to a player.
`level`	Number	Yes	1	Stores the player's current level.
`experiencePoints`	Number	Yes	0	Tracks experience points gained by the player.
`score`	Number	Yes	0	Player's total score.
`rank`	Number	No	No	Player's leaderboard ranking (optional).
`achievements`	[String]	No	[]	Stores unlocked achievement names.
`progress`	String	No	"Not started"	Describes current game progress (e.g., "`Level 3 - Boss Battle`").
`lastPlayed`	Date	No	`Date.now`	Stores the last date the player played.
`updatedAt`	Date (auto-generated)	No	Auto	Timestamp for when the document was last updated.

II. Initial Frontend Development

1. **Authentication Micro Frontend -** Develop a micro frontend using React Vite and Apollo Client.
   a. Create a micro frontend responsible for user authentication (signup, login, logout).
   b. Integrate with the Authentication Microservice for user-related operations.

2. **Game Progress Micro Frontend -** Develop a micro frontend using React Vite and Apollo Client.
   a. Create a micro frontend responsible for:
      - **Leaderboards** to display player rankings dynamically.
      - **Achievements** to showcase earned badges and milestones.
      - **Game progress tracking** with real-time updates.

   b. Integrate with the Game Progress Microservice Game progress tracking with real-time updates.
   c. **Enhance user experience with Three.js** by incorporating **visual effects**, such as:
      - **3D animations** for leaderboard transitions and progress tracking.
      - **Interactive 3D elements** for achievement unlocks (e.g., spinning trophies, glowing badges).
      - **Smooth animations** when updating game progress to create an engaging interface.

Use **functional components**, client-side composition, and **React Hooks** for the Micro Frontends. Design a visually appealing and user-friendly UI.

(10 marks)

Evaluation:

Functionality(including code explanation during demonstration):	
Micro frontends (Authentication, Game Progress**)**	30%

MongoDB database (config files, models)	5%
GraphQL Microservices (Authentication, Game Progress)	30%
Integration using Vite Module Federation plugin	20%
Friendliness (using CSS to align the React elements, React-Bootstrap, etc.)	5%
Code **demonstration** and brief explanation during demonstration in class	10%
Total	**100%**

VS Code Project Naming rules:

You must name your **VS Code** project/folder according to the following rule:
YourFullName_COMP308LabNumber_ ExNumber.

Example: **JohnSmith_JaneSmith_COMP308Lab3_ Ex1**

Submission rules:

Remove the node_modules folder before zipping the project. Submit your project as a **zip file** that is named according to the following rule:
YourFullName _COMP308LabNumber_ExNumber.zip

Example: **JohnSmith_JaneSmith_COMP308Lab3_Ex1.zip**

DO NOT use RAR or other types of archives.