**GITHUB**

git remote add origin <https://github.com/aimanmrkhan/Generative-AI-for-Software-Development>

Github token ghp\_IaZvmCTs7DnccanPgLWbHwdORLnFu24KnhEm

**mangodb**

username khanmohamme4

password: MTk4Ni1raGFubW9o

Secrete key : 628a3118e041c3ce1fe86bfe86fcb11f56011b61baf9812b1d3cb2231a53bab8

https://labs.cognitiveclass.ai/v2/tools/cloud-ide-kubernetes?ulid=ulid-79755a063b27e5a3d2949c5fa88d61a30600c005

I want to create a personalized online learning platform. I want to start with the server side. Recommend a good design and architecture for the server side and help me understand what is required.

You will need a robust and scalable server-side architecture to create a personalized online learning platform. One popular design is a microservices architecture. This architecture divides the application into smaller, independent services that you can develop, deploy, and scale individually.

Here are some key components and requirements for the server side of your learning platform:

1. **Authentication and authorization:** Implement a secure user authentication system to ensure that only authorized users can access the platform. Some examples of a secure user authentication system are OAuth or JSON Web Tokens (JWT).
2. **User management:** Develop features to manage user profiles, including registration, login, password reset, and profile customization.
3. **Content management:** Create a system to manage and organize learning content, such as courses, lessons, quizzes, and multimedia resources. You may consider using a content management system (CMS) or a database to store and retrieve this information.
4. **Learning management system (LMS):** Implement features to track and manage learners' progress, including course enrollment, completion tracking, and assessment management.
5. **Analytics and reporting:** Incorporate analytics tools to gather data on user activities, course performance, and engagement metrics. This data can help you improve the platform and personalize the learning experience.
6. **Payment integration:** To offer paid courses or subscriptions, integrate a secure payment gateway to handle transactions and manage user subscriptions.
7. **Scalability and performance:** Design your server-side architecture to handle concurrent users and ensure quick response times. Consider using cloud-based solutions like AWS or Azure to scale your infrastructure as needed.
8. **API development:** Create a well-documented API that allows integration with other systems, such as mobile apps or third-party services.

Remember to prioritize security, scalability, and performance while designing your server-side architecture. Additionally, keep an eye on the latest trends and technologies in the e-learning industry to stay up-to-date with the best practices.

I would like to use a microservices architecture for the server side. These are the services I want to be able to provide.

Personalized learning recommendations,

Interactive coding exercises

Real-time feedback to help developers improve their skills and knowledge.

What are the various components I should have?

To implement a microservices architecture for your server side, you can consider the following components for the services you want to provide:

1. **User management service:** This service handles user registration, authentication, and authorization. It manages user profiles, preferences, and access control.
2. **Recommendation service:** This service analyzes user data, such as past learning activities, preferences, and performance, to provide personalized learning recommendations. It can use machine learning algorithms to suggest relevant courses, modules, or resources.
3. **Content management service:** This service manages the storage, organization, and retrieval of learning materials, such as videos, documents, or interactive coding exercises. It ensures efficient content delivery and version control.
4. **Exercise service:** This service provides interactive coding exercises for developers. It can support various programming languages and frameworks, provide code execution environments, and offer real-time code correctness and performance feedback.
5. **Feedback service:** This service collects and analyzes user feedback on coding exercises and learning materials. It provides real-time feedback, suggestions, and guidance to help developers improve their skills and knowledge.
6. **Analytics service:** This service tracks user behavior, learning progress, and performance metrics. It generates reports and insights to evaluate the platform's effectiveness, identify improvement areas, and personalize learning experiences further.
7. **Notification service:** This service handles notifications and communication with users. It can send reminders, updates, or notifications about new learning resources, achievements, or personalized recommendations.
8. **Integration service:** This service integrates with external systems, such as authentication providers, content delivery networks, or third-party APIs. It ensures smooth data exchange, interoperability, and seamless user experiences.

Remember to design each microservice independently, loosely coupled, and scalable. Each service should have its own API, data storage, and business logic. Use synchronous or asynchronous protocols, such as RESTful APIs or message queues, for communication between services.

Additionally, consider implementing service discovery mechanisms, centralized logging, monitoring, and error-handling strategies to ensure system reliability, performance, and fault tolerance.

These components provide a starting point for your microservices architecture. You can refine and expand them based on your specific requirements and constraints.

I would like to create the user service. I would like to use Node.js and MongoDB for this project. How do I create a project structure?

To create a project structure for your user management service using Node.js and MongoDB, you can follow these steps:

1. **Set up a new Node.js project:** Open your terminal or command prompt and navigate to the intended directory where you want to create your project. Run the command mkdir user-management-service to create a new directory for your project. Then, navigate into the project directory by running cd user-management-service.
2. **Initialize a new Node.js project:** Run the command npm init to initialize a new Node.js project. Follow the prompts to set up your project details, such as the project name, version, description, and so on. This initialization will create a package.json file in your project directory.
3. **Install required dependencies:** Run the command npm install express mongoose bcrypt jsonwebtoken to install the required dependencies for your user management service. Express is a popular web framework for Node.js. Mongoose is an object modeling tool for MongoDB. Bcrypt is a library for password hashing. The JSON web token is a library for handling JSON Web Tokens for authentication.
4. **Create the project structure:** In your project directory, create the following directories and files:
   * src directory will contain your source code.
   * src/controllers directory will contain your user-related controller logic.
   * src/models directory will contain your user-related database models.
   * src/routes directory will contain your user-related API routes.
   * src/config directory will contain your configuration files.
   * src/index.js file will serve as the entry point for your application.
5. **Implement the User Management Service:** Create the necessary files and folders within the src directory. For example, you can create src/controllers/userController.js, src/models/userModel.js, and src/routes/userRoutes.js to handle user-related operations.
   * In userController.js, define functions for user registration, login, profile management, and so on.
   * In userModel.js, define the user schema using Mongoose to define the user data structure.
   * In userRoutes.js, define the API routes for user registration, login, profile management, and so on.
6. **Configure the MongoDB connection:** In the src/config directory, create a file called database.js to configure the MongoDB connection. Use the Mongoose library to connect to your MongoDB database.
7. **Set up the Express server:** In index.js, import the necessary dependencies, configure the Express server, and set up the API routes.
8. **Test the User Management Service:** Run the command node src/index.js to start the Express server and test your User Management Service using tools like Postman or by making API requests from your frontend application.

Remember to follow best practices for code organization, error handling, and security when developing your User Management Service.

Here's an example of the code you can include in each of the files for your user management service:

1. src/controllers/userController.js:
2. const User = require('../models/userModel');
3. const bcrypt = require('bcrypt');
4. const jwt = require('jsonwebtoken');
5. // User registration
6. exports.registerUser = async (req, res) => {
7. try {
8. const { username, password } = req.body;
9. // Check if the username already exists
10. const existingUser = await User.findOne({ username });
11. if (existingUser) {
12. return res.status(409).json({ message: 'Username already exists' });
13. }
14. // Hash the password
15. const hashedPassword = await bcrypt.hash(password, 10);
16. // Create a new user
17. const newUser = new User({ username, password: hashedPassword });
18. await newUser.save();
19. return res.status(201).json({ message: 'User registered successfully' });
20. } catch (error) {
21. return res.status(500).json({ message: 'Internal server error' });
22. }
23. };
24. // User login
25. exports.loginUser = async (req, res) => {
26. try {
27. const { username, password } = req.body;
28. // Check if the username exists
29. const existingUser = await User.findOne({ username });
30. if (!existingUser) {
31. return res.status(401).json({ message: 'Invalid username or password' });
32. }
33. // Check if the password is correct
34. const isPasswordCorrect = await bcrypt.compare(password, existingUser.password);
35. if (!isPasswordCorrect) {
36. return res.status(401).json({ message: 'Invalid username or password' });
37. }
38. // Generate a JSON Web Token (JWT)
39. const token = jwt.sign({ username: existingUser.username }, 'your-secret-key', { expiresIn: '1h' });
40. return res.status(200).json({ token });
41. } catch (error) {
42. return res.status(500).json({ message: 'Internal server error' });
43. }
44. };
45. // User profile management
46. exports.updateUserProfile = async (req, res) => {
47. try {
48. const { username } = req.params;
49. const { newUsername } = req.body;
50. // Update the user's username
51. await User.updateOne({ username }, { username: newUsername });
52. return res.status(200).json({ message: 'User profile updated successfully' });
53. } catch (error) {
54. return res.status(500).json({ message: 'Internal server error' });
55. }
56. };
57. 1
58. src/models/userModel.js:
59. const mongoose = require('mongoose');
60. const userSchema = new mongoose.Schema({
61. username: {
62. type: String,
63. required: true,
64. unique: true,
65. },
66. password: {
67. type: String,
68. required: true,
69. },
70. });
71. const User = mongoose.model('User', userSchema);
72. module.exports = User;
73. src/routes/userRoutes.js:
74. const express = require('express');
75. const router = express.Router();
76. const userController = require('../controllers/userController');
77. router.post('/register', userController.registerUser);
78. router.post('/login', userController.loginUser);
79. router.put('/:username', userController.updateUserProfile);
80. module.exports = router;

src/config/database.js

1. const mongoose = require('mongoose');
2. const User = require('../models/userModel');
3. const connectDB = async () => {
4. try {
5. await mongoose.connect('mongodb://root:password@localhost:27017');
6. console.log('MongoDB connected');
7. } catch (error) {
8. console.error('MongoDB connection error:', error);
9. process.exit(1);
10. }
11. };
12. module.exports = connectDB;

(In place of password place the password copied from exercise done through mangodb)

src/index.js

1. const express = require('express');
2. const connectDB = require('./config/database');
3. const userRoutes = require('./routes/userRoutes');
4. const app = express();
5. // Connect to MongoDB
6. connectDB();
7. // Middleware
8. app.use(express.json());
9. // Routes
10. app.use('/users', userRoutes);
11. // Start the server
12. const port = 3000;
13. app.listen(port, () => {
14. console.log(`Server started on port ${port}`);
15. });

Make sure to replace 'your-secret-key' with your secret key for JSON Web Token signing and 'your-database-name' with the name of your MongoDB database. In the Database.js file, update the 'password' obtained when starting the MongoDB server.

***ote****:*

1. To obtain the secret key, execute the following command in the terminal.

node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"

*Disclaimer: Your response might vary.*

# Test the application

To run and test your user management service, you can follow these steps:

1. To start your MongoDB server, refer to the pre-work lab for the final project.
2. Start your Node.js server: In your project directory, open a new terminal or command prompt window and run the command node src/index.js to start your Node.js server.

### Generating a database to test the application

You have the code now but you have not created the database yet. You will now use Generative AI to populate the database.

1. Here Can you please provide the user data in JSON format?

is an example of user data in JSON {

1. "name": "John Doe",
2. "password": "password123"
3. },
4. {
5. "name": "Jane Smith",
6. "password": "password456"
7. }
8. ]

You can also verify the endpoint by using a curl command in the terminal window.

curl -X POST -H "Content-Type: application/json" -d '{"username": "john", "password": "password123"}' http://localhost:3000/users/login

You can verify the endpoint using a curl command in the terminal window.

1. 1
2. curl -X PUT -H "Content-Type: application/json" -d '{"newUsername": "john\_smith"}' http://localhost:3000/users/john\_smith

You must provide the code in each file you created and get them reviewed.

1. 1

# Code review

1. Can you review the code below?

And then paste the code that you want to get reviewed.

# Documentation

You need to provide documentation and comments for all the code written.

1. 1
2. Can you provide documentation and comments for the code to make it readable?

Copied!

You will use the prompt iteratively with the content of each file.

You have to iteratively give the code in each file you have created and get them reviewed.

1. Kindly review the userRoutes.js file code.
2. const express = require('express');
3. const router = express.Router();
4. const userController = require('../controllers/userController');
5. router.post('/register', userController.registerUser);
6. router.post('/login', userController.loginUser);
7. router.put('/:username', userController.updateUserProfile);
8. module.exports = router;

The code you provided in the userRoutes.js file looks correct and follows the same structure as the previous code snippet. It imports the necessary modules, sets up the router, and defines the routes for user registration, login, and profile updates.

One thing to note is that the code assumes that the userController module is located in the **../controllers/userController** file relative to the userRoutes.js file. Ensure the file path is correct and the userController module is defined in that file.

The code should work as expected to handle user-related operations on your learning platform.

# Dockerfile

You need to bundle the application in Docker. Type the following prompt to create a Dockerfile that bundles the application and MongoDB server in a container.

Go to the dirctory where package.json is there…

Create a file Docker

1. # Use an official Node.js runtime as a parent image
2. FROM node:14
3. # Set the working directory in the container
4. WORKDIR /usr/src/app
5. # Copy package.json and package-lock.json to the working directory
6. COPY package\*.json ./
7. # Install app dependencies
8. RUN npm install
9. # Bundle app source
10. COPY . .
11. # Expose the port the app runs on
12. EXPOSE 3000
13. # Define the command to run the app
14. CMD ["npm", "start"]

Create another file

docker-compose.yml script in the same directory as the Dockerfile. Here's an example of what the script could look like:

1. version: '3'
2. services:
3. mongo:
4. image: mongo
5. ports:
6. - "27018:27017"
7. app:
8. build: .
9. ports:
10. - "3005:3005"
11. depends\_on:
12. - mongo

To build and run the Docker image, follow these steps:

*Note: Ensure that your Node.js server and MongoDB server are running.*

the Docker image: Open a terminal or command prompt window, navigate to the Dockerfile's directory, and run the command docker build -t your-image-name

Run the Docker container: After building the image, you can run the container using the command docker run -p 3000:3000 -d your-image-name command. This action will start the container in detached mode and map port 3000 of the container to port 3000 of the host machine.

You should now be able to access your Node.js application running inside the Docker container at http://localhost:3000. The MongoDB server also runs inside the container and can be accessed from the Node.js application using the MONGO\_HOST environment variable.

Remember to replace 'your-image-name' with the name for your Docker image.

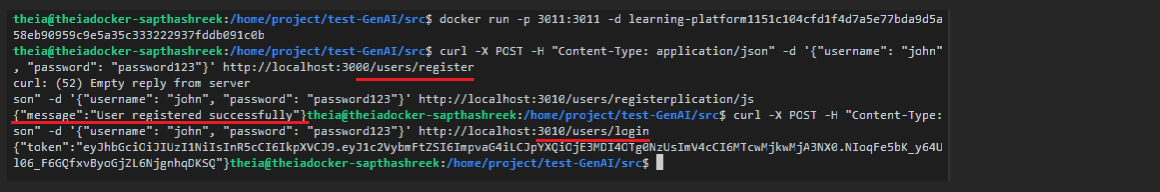
'your-image-name' =user-management-service

**Docker**

docker build -t user-management-service .

docker run -p 3005:3005 -d user-management-service

You can verify the endpoints using CURL commands

**

# Checklist

At this stage:

1. You now have a running application that offers CRUD microservices for the User Management Service.
2. The code has undergone a thorough review and is comprehensively documented.
3. The service has been successfully deployed within a Docker container.
4. Proceed to push all the code to your GitHub repository.

## Summary

* You have successfully gathered requirements for a user management service of a programming-focused learning platform using Generative AI.
* You have explored vital aspects such as fundamental features, user-friendly design, interactive functionalities, and an efficient folder structure.

Subsequent actions involve:

* Employing MongoDB for user data
* Producing Node, Express, and Mongoose code
* Conducting a detailed code review with comprehensive documentation

**Congratulations! You have successfully leveraged Generative AI to build a learning platform by choosing Microservices Architecture, Node.js, and MongoDB.**

Click here to view the steps to push your work to GitHub

Remember, any modifications you make in the lab environment won't be saved. If you plan to step away, use the following steps to ensure your changes push to GitHub:

Ensure that you are currently inside the Project directory.

1. Navigate to the project directory by using: cd <project dir>.
2. Set up your Git configuration:
   * Run: git config --global user.email "[yourgithub@email.com](mailto:yourgithub@email.com)"
   * Run: git config --global user.name "name"
3. Add your changes to the staging area:
   * Run: git add .
4. Commit your changes with a descriptive message:
   * Run: git commit -m "Adding temporary changes to Github"
5. The first step is to generate an access token from GitHub.com. Follow directions from the lab [Generate GitHub personal access token](https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0131EN-SkillsNetwork/labs/create-personal-token/instructions.md.html) and copy the access token as a password in the upcoming exercises.
6. Push your changes to the Git repository:
   * Run: git push
7. A prompt in the terminal will prompt you to enter your GitHub username and password (your previously created Personal Access Token from Step 5).

These steps ensure your work is safely stored in GitHub, allowing you to continue when you return to the lab environment.