Cloud Computing Coursework

1. Development of a Cloud Software as a Service

The coursework aims to make you apply the concepts and software development methods and frameworks seen in class to develop a Cloud Software as a Service (SaaS).

♦ You will need to install, develop, test and deploy a Cloud SaaS in virtualised environments, such as VMs, containers and Kubernetes, as per the guidelines described in this document.

♦ Upload your final scripts and a technical report to describe the functionality of your solution following the guidelines of the coursework brief document.

2. The Piazza system

You must develop a RESTful SaaS for a Twitter-like system called **Piazza**. In **Piazza**, users post messages for a particular topic while others browse posts, e.g., per topic and perform fundamental interactions, including liking, disliking, or adding a comment. You should install, test, and document your developments as described in Section 3.

Piazza should support the following actions.

Action 1:	Authorised users access the **Piazza** API using the oAuth v2 protocol to perform any interaction.
Action 2:	Authorised users post a message for a particular topic in the **Piazza** API.
Action 3:	Registered users browse messages per topic using the **Piazza** API.
Action 4:	Registered users perform basic operations, including "like", "dislike", or "comment" a message posted for a topic.
Action 5:	Authorised users could browse for the most active post per topic with the highest likes and dislikes.
Action 6:	Authorised users could browse the history data of expired posts per topic.

You recommended using the Node.js Express package and MongoDB to develop your software. If you like, you can use any other programming framework or database you prefer. You are advised to use code samples from the lab tutorials and online sources by providing the appropriate references (in text and code). In the technical report, you can improvise by describing your functionality and implementation decisions and include a diagram to demonstrate your implementation.

3. Coursework phases

Phase A: Install and deploy software in virtualised environments [5 marks]

- Install all the necessary packages in your virtual machine.
- Deploy your code in the virtual machine using your GitHub repository.
- Your REST API endpoints should be available under your virtual machine IP address based on examples and guidelines seen in class.
- Provide a short description of your setup in the report.
- Briefly discuss your installation and the structure of your folders.
  - Provide screenshots if necessary.
- Do not include Linux commands in the report.

Phase B: Enforcing authentication/verification functionalities [5 marks]

- Your service should include user management and JWT functionality using NodeJS.
  - The JWT will authorise the auctioning RESTful API to store data for authorised users in your database.
  - You can use a MongoDB database for storing data required in the coursework, as shown in the labs.
- Your code should authenticate users each time you perform any action point of the following Part C, for example, whenever users post, interact, or browse messages.
  - Unauthorised users are not allowed to access your resources and perform database requests.
- Any user input should follow a verification process for validation purposes. You are encouraged to improvise and apply your validations.

Phase C: Development of Piazza RESTful APIs [25 marks]

- Your APIs should allow the basic functionalities provided in the Action points of Section 2.
- Each post on the **Piazza** wall should include the following data:
  - The post identifier.
  - The title of a post.
  - The topic of a post from one of the following four categories: Politics, Health, Sport or Tech. Each post could belong to one or more topics.
  - A timestamp of the post-registration in the **Piazza** API.
  - The message body of a post.

- ◆ The post-expiration time. After the end of this time, the message will remain on the **Piazza** wall, but it will not accept any further actions, e.g. (likes, dislikes, or comments).
  - ◆ The status of a post could be "Live" or "Expired". Is the status automatically set once expiry date is equal to today?
  - ◆ Information about the post owner (e.g., a name).
  - ◆ The number of likes, dislikes, or a list of comments, if any.
  - ◆ Any other information you might need to store essential for your project.
- Each user interaction with a post on a topic (like, dislike or comment) should include the following data:
  - ◆ Information about the user interacting with the post of a topic (e.g., a name).
  - ◆ The interaction value (including a like, a dislike, or a comment made).
  - ◆ The time left for a post to expire. Is that calculated when the user interacts?
  - ◆ Any other information you can store essential for your project.
- You are encouraged to develop your database collections and application logic and expand Phase C as required.
  - ◆ Provide a brief description of your database models in the report.
  - ◆ Use screenshots if necessary.
- You should list the RESTful API endpoints in your report with a simple example of using your API.
- Consider the following.
  - ◆ Only authorised users should access the API.
  - ◆ The API should allow any registered user to perform posts.
  - ◆ You are encouraged to set any other constraints to improve the software's functionality.
  - ◆ You are encouraged to improvise and develop any functionality you like.

Phase D: Testing your application [15 marks]

You can demonstrate the testing aspects of your software using Postman, e.g., by including screenshots in the report or by developing a sting application. For example, you can develop a testing application using Node.js or Python to automate the test cases of users posting messages. This application should be developed outside your virtual machine, e.g., on your desktop or laptop. The testing application should connect to the API endpoints and perform the required HTTP calls to manipulate data. The test cases should demonstrate Action points, as described in Section 2. The test cases will verify your system functionality.

Let us assume a use case scenario of four users, Olga, Nick, Mary, and Nestor, accessing the Piazza API. Provide the following test cases (TCs).

TC 1. Olga, Nick, Mary, and Nestor register and are ready to access the **Piazza** API.

TC 2. Olga, Nick, Mary, and Nestor use the oAuth v2 authorisation service to register and get their tokens.

TC 3. Olga makes a call to the API without using her token. This call should be unsuccessful as the user is unauthorised.

TC 4. Olga posts a message in the Tech topic with an expiration time (e.g. 5 minutes) using her token. After the end of the expiration time, the message will not accept any further user interactions (likes, dislikes, or comments).

TC 5. Nick posts a message in the Tech topic with an expiration time using his token.

TC 6. Mary posts a message in the Tech topic with an expiration time using her token.

TC 7. Nick and Olga browse all the available posts in the Tech topic; three posts should have zero likes, zero dislikes, and no comments.

TC 8. Nick and Olga "like" Mary's post on the Tech topic.

TC 9. Nestor "likes" Nick's post and "dislikes" Mary's on the Tech topic.

TC 10. Nick browses all the available posts on the Tech topic; at this stage, he can see the number of likes and dislikes for each post (Mary has two likes and one dislike, and Nick has one like). No comments have been made yet.

TC 11. Mary likes her post on the Tech topic. This call should be unsuccessful; in **Piazza,** a post owner cannot like their messages.

TC 12. Nick and Olga comment on Mary's post on the Tech topic in a round-robin fashion (one after the other, adding at least two comments each). *how many times?*

TC 13. Nick browses all the available posts in the Tech topic; at this stage, he can see the number of likes and dislikes of each post and the comments made.

Is the expiration time randomly set or is it based on the token?

TC 14. Nestor posts a message on the Health topic with an expiration time using her token.

TC 15. Mary browses all the available posts on the Health topic; at this stage, she can see only Nestor's post.

TC 16. Mary posts a comment in Nestor's message on the Health topic.

TC 17. Mary dislikes Nestor's message on the Health topic after the end of post-expiration time. This should fail.

TC 18. Nestor browses all the messages on the Health topic. There should be only one post (his own) with one comment (Mary's).

does she choose to browse the expired messages or is it that the messages are expired?

TC 19. Nick browses all the expired messages on the Sports topic. These should be empty.

TC 20. Nestor queries for an active post with the highest interest (maximum number of likes and dislikes) in the Tech topic. This should be Mary's post.

Feel free to develop more test cases to test your implementations. You should provide screenshots or descriptions of each test case in the report.

Phase E: Deploy your Piazza project into a VM using Docker [10 marks]

Upload your code to a GitHub repo and then deploy it in a Google Cloud VM. Provide a list of commands in the report and screenshots to demonstrate your deployment actions. As an alternative, you can use DockerHub.

Phase F: Deploy your application in Kubernetes [15 marks]

Create a Kubernetes cluster on the Google Cloud platform and deploy your application. You can use the DockerHub to move your code or follow a manual approach. Then follow the next specification:

- Deploy a load balancer following the specifications seen in class.
- Deploy the Piazza application by including five replicas of your service.

Phase G: Present your solution in a technical report [20 marks]

Provide details on your implementations, database design, service descriptions, API resources, and any other information required. You do not need to include a list of the Linux commands in the report. Provide references to online resources, including source codes from online tutorials or repositories. There is no report size, but you should keep the report short and solid regarding the development phases.

Phase H: Submit quality scripts [5 marks]

Your codes should be written in such a way that makes them highly readable and easy to follow. Good use of REST concepts, understanding of computational complexity in software development, use of comments in code, proper database models, proper indentation, clear notations, and simplicity in the flow are factors to consider.

Phase I: Go beyond the coursework requirements [0 marks]

Feel free to create your application name, and don't forget to design your logo! Improvise and expand the Piazza functionalities. Using CI/CD pipelines with Terraform to deploy or develop a front-end could highlight some of these areas.

4. General coursework guidelines

Consider the following when developing your software.

- You should provide solutions using Node.js, MongoDB and Postman to test your endpoints.
- As seen in class, you are encouraged to reuse the lab tutorials to deploy and develop software.

- Follow the instructions of the coursework brief and coursework description on the Moodle page for the time and mode of submission of your software.
- Provide comments to explain key functionalities and implementations.
- Whenever necessary, provide screenshots of your API calls to demonstrate functionality.
- Provide a clear description and example of your API endpoints in the report.
- There is no limit to the report size.
- Provide clear explanations of the test cases and future work.
- The coursework requires you to create database tables in MongoDB to save your data so you can improvise as needed.
- Consider that there could be multiple ways to implement tables to manage the data of your developments.

As I work on the project, I need to report progress step by step and remain focused on maintaining code quality and following best practices.

Assignment Marking Criteria

Part of assessment	Criteria	Marks
Phase A: Install and deploy software in virtualised environments	a) Install all the necessary packages in your virtual machine. b) Deploy your code in the virtual machine using your GitHub repository. c) REST API endpoints should be available under your virtual machine IP address based on examples and guidelines seen in class. Provide screenshots. d) Provide a short description of your setup in the report. e) Discussion of installation and the structure of your folders.	5
Phase B: Enforcing authentication/verification functionalities	a) User management and JWT functionality using NodeJS. b) Authenticate users each time you perform any action point. c) Complete verification process for validation purposes.	5
Phase C: Development of Piazza RESTful APIs	a) Implement basic functionalities provided in the Action points of Section 2. b) Complete requirements as discussed in Phase C section.	25
Phase D: Testing your application	a) Demonstrate testing of application by either using screenshots of postman, or by developing a testing application e.g., using Node.js or Python to test endpoints, or by any other use of testing libraries.	15

isn't that already part of the report?

	b) Discuss and document test cases either by including screenshots or code samples. c) Complete implementation of the twenty use cases as presented in section Phase D.	
Phase E: Deploy your Piazza project into a VM using Docker	Upload code to a GitHub repository and then deploy it in a Google Cloud VM. Provide a list of commands in the report and screenshots to demonstrate your deployment actions. As an alternative, DockerHub can be used.	10
Phase F: Deploy your application in Kubernetes	a) Create a Kubernetes cluster on the Google Cloud platform and deploy your application. b) Complete requirements for load balancing and replication as discussed in section Phase F.	15
Phase G: Report your solution in a technical report	a) Provide details on your implementations, database design, service descriptions, API resources, and any other information required. b) Provide references using the Harvard referencing system as needed in the report	20
Phase H: Submit quality scripts	Code samples should be written in such a way that makes them highly readable and easy to follow. Good use of REST concepts, understanding of computational complexity in software development, use of comments in code, proper database models, proper indentation, clear notations, and simplicity in the flow are factors to consider.	5
		100