


Assignment #02

Due Date

Section	Submission Due Date	Grading Due Date
Boston	01/31/2024 @ 9 PM EDT	02/07/2024 @ 9 PM EDT
Seattle & Silicon Valley	02/02/2024 @ 9 PM PDT	02/09/2024 @ 6 PM PDT

Getting Help

 **Info**

Ask all your questions in Canvas Discussion. SEA/SJO can ask questions in Teams.

Learning Objectives

The objective of this assignment is to select a technology stack for a backend (API only) web application and to implement health check API. The technology stack must meets [Cloud-Native Web Application Requirements](#) as outlined below.

Cloud Native Web Application Requirements

A cloud-native application is an application that is specifically designed for cloud computing architecture. It takes advantage of cloud computing frameworks, which are composed of loosely coupled cloud services.

- **Server Operating System:** CentOS 8
- **Programming Language:** Any high-level programming language such as Java, Python, Go, Node.js or PHP.
- **Relational Database:** Either *MySQL* or *PostgreSQL*. Do not use MongoDB, MSSQL, Oracle, etc.
- **Backend Framework:** Any open-source framework such as Spring, Hibernate, etc.

- **ORM Framework:**
 - **Java:** [Hibernate](#)
 - **Golang:** [GORM](#)
 - **JavaScript:** [Sequelize](#)
 - **Python:** [SQLAlchemy](#)
- **UI Framework:** N/A
- **CSS:** N/A

Web Application Development



RESTful API Requirements

1. All API request/response payloads should be in JSON.
2. No UI should be implemented for the application.
3. As a user, I expect all API calls to return with a proper [HTTP status code](#).
4. As a user, I expect the code quality of the application to be maintained to the highest standards using the unit and/or integration tests.



Health Check RESTful API

- The question we want to answer is `How to detect that a running service instance is unable to handle requests?`.
- The health check API is a way for us to monitor the health of the application instance and alerts us when something is not working as expected.
- Health check API allows us to stop sending traffic to unhealthy instances of the application and to automatically replace/repair them. It also helps us improve user experience by not routing their requests to unhealthy instances.

A Health Check API may check for following:

1. **Database connection** - Make sure the application is connected to the database or is able to establish connection at the time of the health check.
2. **Downstream API Calls** - Your application may depend on other downstream APIs and outage of downstream API without which you cannot complete the users request.

For this assignment, we are going to implement an endpoint `/healthz` that will do the following when called:

1. Check if the application has connectivity to the database.
 - a. Return `HTTP 200 OK` if the connection is `successful`.
 - b. Return `HTTP 503 Service Unavailable` if the connection is `unsuccessful`.
2. The API response should not be cached. Make sure to add `cache-control: 'no-cache'` header to the response.
3. **The API request should not allow for any payload.** The response code should be `400 Bad Request` if the request includes any payload.
4. **The API response should not include any payload.**
5. Only HTTP `GET` method is supported for the `/healthz` endpoint.

Example Requests

Success

```
1 curl -vvvv http://localhost:8080/healthz
2 * Trying 127.0.0.1:8080...
3 * Connected to localhost (127.0.0.1) port 8080 (#0)
4 > GET /healthz HTTP/1.1
5 > Host: localhost:8080
6 > User-Agent: curl/8.1.2
7 > Accept: */*
8 >
9 < HTTP/1.1 200 OK
10 < Cache-Control: no-cache, no-store, must-revalidate;
11 < Pragma: no-cache
12 < X-Content-Type-Options: nosniff
13 < Date: Wed, 20 Sep 2023 01:18:37 GMT
14 < Content-Length: 0
15 <
16 * Connection #0 to host localhost left intact
```

Failure

```
1 $ curl -vvvv http://localhost:8080/healthz
2 * Trying 127.0.0.1:8080...
3 * Connected to localhost (127.0.0.1) port 8080 (#0)
4 > GET /healthz HTTP/1.1
```

```
5 > Host: localhost:8080
6 > User-Agent: curl/8.1.2
7 > Accept: */*
8 >
9 < HTTP/1.1 503 Service Unavailable
10 < Cache-Control: no-cache, no-store, must-revalidate;
11 < Pragma: no-cache
12 < X-Content-Type-Options: nosniff
13 < Date: Wed, 20 Sep 2023 01:18:57 GMT
14 < Content-Length: 0
15 <
16 * Connection #0 to host localhost left intact
```

405 Method Not Allowed

```
1 $ curl -vvvv -XPUT http://localhost:8080/healthz
2 * Trying 127.0.0.1:8080...
3 * Connected to localhost (127.0.0.1) port 8080 (#0)
4 > PUT /healthz HTTP/1.1
5 > Host: localhost:8080
6 > User-Agent: curl/8.1.2
7 > Accept: */*
8 >
9 < HTTP/1.1 405 Method Not Allowed
10 < Cache-Control: no-cache, no-store, must-revalidate;
11 < Pragma: no-cache
12 < X-Content-Type-Options: nosniff
13 < Date: Wed, 20 Sep 2023 01:38:11 GMT
14 < Content-Length: 0
15 <
16 * Connection #0 to host localhost left intact
```

Submission



Warning

The assignment will be considered late if uploaded to Canvas after the due date.

1. Create a folder with the naming convention `firstname_lastname_neuid_##` where `##` is the assignment number.
2. Copy complete code for the assignment into this folder.
3. Create a create a zip of the `firstname_lastname_neuid_##` directory. The zip file should be `firstname_lastname_neuid_##.zip`.
4. Now unzip the zip file in some other directory and confirm the content of the zip files.
5. Upload the Zip to the correct assignment in Canvas.

6. You are allowed to resubmit. If you think there may be an issue with the ZIP file, feel free to submit it again. Only the latest submission will be graded.

Grading Guidelines



Warning

Following guidelines are for information only. They are subject to change at the discretion of the instructor and TA.

Web Application (100%)

- Students will demo the web application from their laptops. Download the zip uploaded to Canvas and use the submitted code for demo. Local code cannot be used for demo.
- APIs can be demoed using any Postman or Restlet or some other REST client but not via the browser.
- The application should not have UI.
- Verify the success and failure by shutting down the database server while the application is still running. Restarting the database without restarting the application should work.
- Verify that the only HTTP request method supported is `GET`. Making `POST`, `PUT`, `DELETE`, or `PATCH` requests should return HTTP status code `405 Method Not Allowed`.
- Verify that request and response have no payload requirement. The API response should NOT include a body. The request should NOT require any query parameters.
- The application connects to either `MySQL` or `PostgreSQL`. No other database is allowed.
- Verify code is not generated by ChatGPT or Google Bard or GitHub co-pilot.
- Application should not throw `500 Internal Server` errors.
- Application should not require restart between API calls.