Assignment #03

Due Date

Section	Submission Due Date	Grading Due Date
Boston	10/04/2023 @ 9 PM EDT	10/11/2023 @ 9 PM EDT
Seattle & Silicon Valley	10/06/2023 @ 6 PM PDT	10/13/2023 @ 6 PM PDT

Getting Help

Info

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

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.

Bootstrapping Database

- The application is expected to automatically bootstrap the database at startup.
- Bootstrapping creates the schema, tables, indexes, sequences, etc., or updates them if their definition has changed.

- The database cannot be set up manually by running SQL scripts.
- It is highly recommended that you use ORM frameworks such as Hibernate (for java), SQLAlchemy (for python), and Sequelize (for Node.js).

Users & User Accounts

1. The web application will load account information from a `CSV` file from well known location `/opt/user.csv`.
2. The application should load the file at startup and create users based on the information provided in the CSV file.
    a. The application should create new user account if one does not exist.
    b. If the account already exists, no action is required i.e. no updates.
    c. Deletion is not supported.
3. Example CSV file can be downloaded from here.
4. The user's password must be hashed using BCrypt before it is stored in the database.
5. Users should not be able to set values for `account_created` and `account_updated`. Any value provided for these fields must be ignored.

Authentication Requirements

1. The user must provide a basic authentication token when making an API call to the authenticated endpoint.
2. The web application must only support Token-Based authentication and not Session Authentication.

API Implementation

> **Note**
>
> About the field data types in Swagger docs.

In this assignment we are going to implement REST API to allow users to create `assignments`. The API specifications can be found here.

The authenticated user should be able to do the following:

1. Create Assignment
    a. Any user can add a assignment.
    b. Assignment points must be between `1` and `10`.
2. Update Assignment
    a. Only the user who created the assignment can update the assignment.
    b. Users can use either the `PUT` API for updates.
3. Delete Assignment
4. Only the user who created the assignment can delete the assignment.
5. Users should not be able to set values for `assignment_created` and `assignment_updated`. Any value provided for these fields must be ignored.

Git & GitHub

GitHub Subscription

- All students will need to subscribe to the GitHub Team plan.

Create & Setup GitHub Repository

1. Create a new `private` GitHub repository for web applications in the GitHub organization you created.
2. The GitHub repository name must be `webapp`.
3. Update `README.md` in your repository. Your readme file must contain the following:
4. Prerequisites for building and deploying your application locally.
5. Build and Deploy instructions for the web application.
6. Fork the GitHub repository in your namespace. You will do all development work on your fork.
7. All web application code should now be in this repository.
8. Add appropriate `.gitignore` to your repository. A collection of useful `.gitignore` templates can be found here.

GitHub Repository Branch Protection Rules

1. Learn more about protected branches.
2. Implement Branch Protection Rules shown in the image below:

Branch name pattern *

main

Protect matching branches

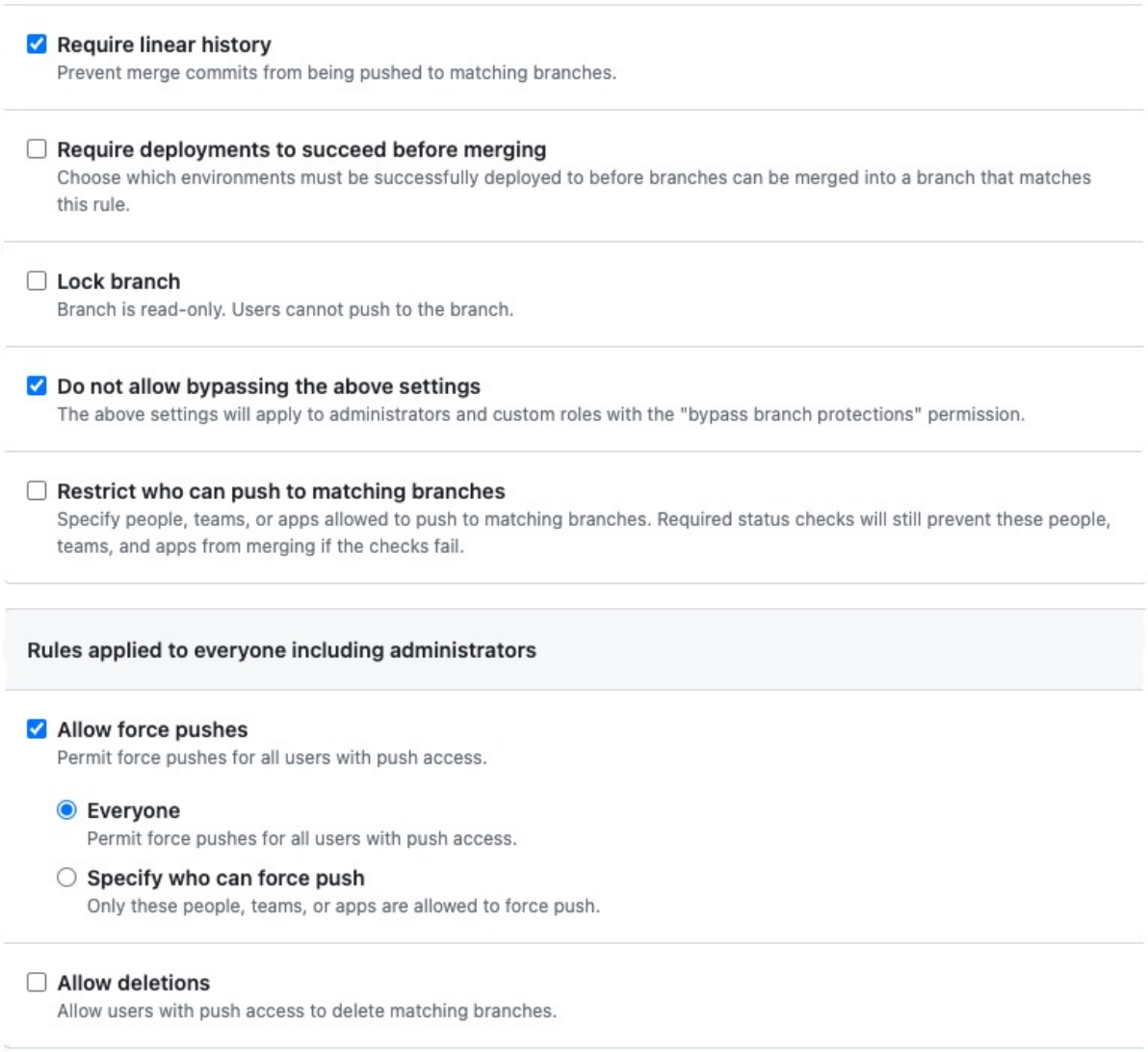
- [x] **Require a pull request before merging**
  When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
  - [ ] **Require approvals**
    When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.
  - [x] **Dismiss stale pull request approvals when new commits are pushed**
    New reviewable commits pushed to a matching branch will dismiss pull request review approvals.
  - [ ] **Require review from Code Owners**
    Require an approved review in pull requests including files with a designated code owner.
  - [ ] **Restrict who can dismiss pull request reviews**
    Specify people, teams, or apps allowed to dismiss pull request reviews.
  - [ ] **Allow specified actors to bypass required pull requests**
    Specify people, teams, or apps who are allowed to bypass required pull requests.
  - [ ] **Require approval of the most recent reviewable push**
    Whether the most recent reviewable push must be approved by someone other than the person who pushed it.
- [x] **Require status checks to pass before merging**
  Choose which status checks must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.
  - [x] **Require branches to be up to date before merging**
    This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

No status checks found

Sorry, we couldn't find any status checks in the last week for this repository.

Learn more about status checks

- [ ] **Require conversation resolution before merging**
  When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. Learn more.
- [ ] **Require signed commits**
  Commits pushed to matching branches must have verified signatures.



Implement Continuous Integration (CI) with GitHub Actions

1. Add a GitHub Actions workflow to run the application `integration` tests for each pull request raised. A pull request can only be merged if the workflow executes successfully.
2. Add Status Checks GitHub branch protection to prevent users from merging a pull request when the GitHub Actions workflow run `fails`.
3. The CI check should run the integration tests you will implement as part of this assignment.

Implement Integration Tests

Implement integration (and not unit) tests for the `/healtz` endpoint. You only need to test for success criteria and not for the failure. This will require your GitHub action to install and setup an

actual `MySQL` and `PostgreSQL` instance and provide configuration to the application to connect to it.

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 Crash (20% Penalty)

- Application should not throw `500 Internal Server` errors.
- Application should not require restart between API calls.

Web Application (60%)

Students will demo the assignment from Debian 12 VM running in Digital Ocean.

Preparing for Demo

1. Launch Debian 12 VM on Digital Ocean (or AWS/GCP).
2. Download code submission from Canvas and `SCP` the zip file to the VM.
3. Install MySQL/MariaDB or PostgreSQL on the VM.
4. Install dependencies such as Node, Python, Java on the VM.
5. Build the application from source in ZIP file if needed.
6. Application configuration may be updated manually at this point.
7. Use the TA provided `user.csv` file and copy it to `/opt/user.csv`.
8. Launch the application and validate that you can hit the `healthz` endpoint and application is healthy.
9. No SQL script should be executed to set up the database.

Web Application Demo

1. Check the database to see if user accounts are created and values for `account_created` and `account_updated` are setup automatically. Since we do not require updates or deletes, both `account_created` and `account_updated` should have same value for now.
2. Check the database to verify passwords are not stored in plain text but instead are hashed using BCrypt algorithm.
3. Application supports basic authentication. Students can demo API calls using Postman or any other REST client.
4. API Testing
    a. Assignment can only be `Created` by authenticated users. `401` should be returned when calls are made without valid `Authorization` headers.
    b. Assignment can only be `Updated` by the user who created it. Others should get `403`.
    c. Assignment can only be `Deleted` by the user who created it. Others should get `403`.
    d. `PATCH` HTTP Method for updating assignments should return `405`.

Git & GitHub Repository (40%)

1. No direct commits are made to organization repository.
2. GitHub repository has branch protection rule setup.
3. GitHub Workflow is triggered when pull request is raised and integration test is executed as part of the workflow. To verify the test, ask student to modify the workflow and remove steps

that installs the database. Health check for the app should fail and integration test itself should fail.

4. Verify student is working from forked repository and using feature branch. There should be no direct commits to their main branch in their forked repository.