Rubric Requirements

Sunday, February 28, 2021 9:25 AM

1. Data Modeling :

Criteria	Meets Specifications	Status	Comments
Architect relational database models in Python	1. Use of correct data types for fields 2. Use of primary and optional foreign key ids	Done	1. Can be verified checking models.py file. 2. The id field in User table is the primary key for the user table and foreign key in Steps table.
Utilize SQLAlchemy to conduct database queries	1. Does not use raw SQL or only where there are not SQLAlchemy equivalent expressions 2. Correctly formats SQLAlchemy to define models 3. Creates methods to serialize model data and helper methods to simplify API behavior such as insert, update and delete.	Done	1. Used SQL Alchemy expressions - can be verified in main.py and auth.py files. 2. Done 3. Helper methods - insert(), update() and delete() created in models.py file and used in main.py and auth.py files

2. API Architecture and Testing

Criteria	Meets Specifications	Status	Comments
Follow RESTful principles of API development	1. RESTful principles are followed throughout the project, including appropriate naming of endpoints, use of HTTP methods GET, POST, and DELETE 2. Routes perform CRUD operations	Done	1. RESTful principles followed and endpoints named to be self-explanatory in the context of the requirement. 2. All routes perform their respective CRUD operations.
Structure endpoints to respond to four HTTP methods, including error handling	1. Specifies endpoints and behavior for at least: a. Two GET requests b. One POST request c. One PATCH request d. One DELETE request 2. Utilize the @app.errorhandler decorator to format error responses as JSON objects for at least four different status codes	Done	1. Endpoints and behavior specified for the below: a. Two GET requests : /user, /allUsers b. One POST request : /addSteps c. One PATCH request : /update d. One DELETE request : /delete 2. Have used blueprint so the errorhandler decorator is @main.errorhandler and @auth.errorhandler . The 4 common status codes covered - 400, 401, 403, 422

Enable Role Based Authentication and roles-based access control (RBAC) in a Flask application	1. Project includes a custom @requires_auth decorator that:
	a. get the Authorization header from the request
	b. Decode and verify the JWT using the Auth0 secret
	c. take an argument to describe the action
	i.e. @require_auth('create:drink')
	d. raise an error if:
	i. the token is expired
	ii. the claims are invalid
	iii. the token is invalid
	iv. the JWT doesn't contain the proper action
	2. Project includes at least two different roles that have distinct permissions for actions. These roles and permissions are
		Done	1. Included @requires_auth decorator which in turn calls the methods :
			a. get_token_auth_header()
			b. verify_decode_jwt(token)
			c. The @requires_auth decorator specifies the permissions required to perform the specific action. For ex:
			`@main.route("/allUsers", methods=['GET', 'POST'])`
			`@requires_auth('get:steps-all')`
			`def all_users_stepsRecords(token):`
			d. Raises appropriate errors for token expiry, invalid claims, invalid token and missing permissions.
			Details can be verified in the files main.py and auth.py
			
			2. Two different roles - User and Admin is created and each of the roles have distinct permissions. User can perform all CRUD operations related to only the specific User and Admin can only view

	clearly defined in the project README. Students can reference the Casting Agency Specs in the Specifications section of this rubric as an example.		only the specific User and Admin can only view steps record of all the Users. The roles and permissions is explained in detail in the README.md file under : **Member Login with Auth0 Authentication and RBAC implementation**
Demonstrate validity of API behavior	1. Includes at least one test for expected success and error behavior for each endpoint using the unittest library 2. Includes tests demonstrating role-based access control, at least two per role.	Done	1. The file test_stepsLogger.py file contains all the tests. Have used the unittest library. 2. There are tests included that demonstrate role-based access control.

3. Third Party Authentication

Criteria	Meets Specifications	Status	Comments
Configure third-party authentication systems	Auth0 is set up and running at the time of submission. All required configuration settings are included in a bash file which export: • The Auth0 Domain Name • The JWT code signing secret • The Auth0 Client ID	Done	Auth0 has been set up. Details are in the config file, have listed below: *configAuth0 = {* *"AUTH0_DOMAIN" : "prisha.au.auth0.com",* *"ALGORITHMS" : ["RS256"],* *"API_AUDIENCE" : "stepsLogger"* *}* Additional details are specified in auth.py file
Configure roles-based access control (RBAC)	• Roles and permission tables are configured in Auth0. • Access of roles is limited. Includes at least two different roles with different permissions. • The JWT includes the RBAC permission claims.	Done	All details are in the README file under the section: **Member Login with Auth0 Authentication and RBAC implementation**

4. Deployment on Heroku:

Criteria	Meets Specifications	Status	Comment
Application is hosted live at student provided URL	• API is hosted live via Heroku • URL is provided in project README • API can be accessed by URL and requires authentication	Done	1. API is hosted via Heroku 2. URL - https://steps-logger.herokuapp.com/ provided in README.md file 3. API can be accessed via URL and authentication is via Member Login
Includes instructions to set up authentication	Instructions are provided in README for setting up authentication so reviewers can test endpoints at live application endpoint	Done	Provided the description and instructions in detail in the README.md file under the sections: 1. Project Description 2. Priya's note to the reviewer wr.r.t. the project 3. Part 1 : Demo of Usage : Signup and User Login Tabs 4. Part 2 : Member Login with Auth0 Authentication and RBAC implementation 5. The reviewer needs to 6. Documentation of API behavior and RBAC controls

5. Code Quality and Documentation

Criteria	Meets Specifications	Status	Comments
Write clear, concise and well documented code	The code adheres to the PEP 8 style guide and follows common best practices, including: 1. Variable and function names are clear. 2. Endpoints are logically named. 3. Code is commented appropriately.	Done	Followed the best practices to meet the specifications. 1. Variable and function names are clear and self-explanatory of the action or purpose. 2. Endpoints are logically named.

	4. Secrets are stored as environment variables.		3. Provided elaborative comments which include the source, if not being taught in the lectures and the logic followed to develop the application 4. Secrets are stored as environment variables primarily in the config.py file. Also in Heroku settings have added the configuration details for the application.
Project demonstrates reliability and testability	1. Application can be run with no errors and responds with the expected results. 2. API test suite for endpoints and RBAC behavior runs without errors or failures	Done	1. Tested both locally and after deploying on Heroku. 2. Run the test_stepsLogger.py and verified all tests pass.
Project demonstrates maintainability	• Variable names are logical, code is DRY and well-commented where code complexity makes them useful	Done	Added elaborative comments in the files: main.py, auth.py, config.py, test_stepsLogger.py, models.py and __init__.py . *Comments that the reviewer needs to pay attention to have been marked as **Priya's note to the reviewer.***
Project includes thorough documentation	• Project includes an informative README a. Motivation for project b. Project dependencies, local development and hosting instructions, c. Detailed instructions for scripts to install any project dependencies, and to run the development server. d. Documentation of API behavior and RBAC controls	Done	Have created the README.md file and furnished the details in the sections: a. **Motivation** b. **Project dependencies, local development and hosting instructions** **c. Detailed instructions for scripts to install any project dependencies, and to run the development server.** **d. Documentation of API behavior and RBAC controls** In addition have **also included** : a. Project Description - at the very beginning of the README.md file b. Links referred - towards the end c. Questions asked in Udacity's knowledge center d. Challenges faced during the development