Portfolio Assignment: Final Project

Purpose

In this assignment you will pull together all of the pieces you have worked on up to this point. You will need to implement a REST API that uses proper resource based URLs, pagination and status codes. In addition you will need to implement some sort of system for creating users and for authorization. You will deploy your application on Google Cloud Platform.

- The default is that you must use
    - Datastore to store your data, and
    - Either Node.js or Python 3, and
    - Google App Engine to deploy your project
- However, contact the instructor if you want to use
    -
        - A different database supported on GCP, or
        - A different programming language, or
        - A different GCP service to deploy your project
    - The instructional staff may not be able to provide much help if you follow this path, but requests to use something different are very likely to be accepted.
    - Note: if you already have permission to use some non-default language, framework or service, you don't need to ask for permission again.

Instructions

Your application needs to have

- An entity to model the user.
- At least two other non-user entities.
- The two non-user entities need to be related to each other.
- The user needs to be related to at least one of the non-user entities.
- Resources corresponding to the non-user entity related to the user must be protected.

Requirements for non-user entities

1. For each entity a collection URL must be provided that is represented by the collection name.
    - E.g., GET /boats represents the boats collection

2. If an entity is related to a user, then the collection URL must show only those entities in the collection which are related to the user corresponding to the valid JWT provided in the request
    - E.g., if each boat is owned by a user, then GET /boats should only show those entities that are owned by the user who is authenticated by the JWT supplied in the request
3. For an entity that is not related to users, the collection URL should show all the entities in the collection.
4. The collection URL for an entity must implement pagination showing 5 entities at a time
    - At a minimum it must have a 'next' link on every page except the last
    - The collection must include a property that indicates how many total items are in the collection
5. Every representation of an entity must have a 'self' link pointing to the canonical representation of that entity
    - This must be a full URL, not relative path
6. Each entity must have at least 3 properties of its own.
    - id and self are not consider a property in this count.
    - Properties to model related entities are also not considered a property in this count.
        - E.g., a boat is not a property of a load in this count, and neither is the owner of a boat.
    - Properties that correspond to creation date and last modified date will be considered towards this count.
7. Every entity must support all 4 CRUD operations, i.e., create/add, read/get, update/edit and delete.
    - You must handle any "side effects" of these operations on an entity to other entities related to the entity.
        - E.g., Recall how you needed to update loads when deleting a boat.
    - Update for an entity should support both PUT and PATCH.
8. Every CRUD operation for an entity related to a user must be protected and require a valid JWT corresponding to the relevant user.
9. You must provide an endpoint to create a relationship and another to remove a relationship between the two non-user entities. It is your design choice to make these endpoints protected or unprotected.
    - E.g., In Assignment 4, you had provided an endpoint to put a load on a boat, and another endpoint to remove a load from a boat.
10. If an entity has a relationship with other entities, then this info must be displayed in the representation of the entity
    - E.g., if a load is on a boat, then
        - The representation of the boat must show the relationship with this load

        - The representation of this load must show the relationship with this boat
11. There is no requirement to provide dedicated endpoints to view just the relationship
    - E.g., Assignment 4 required an endpoint /boats/:boat_id/loads. Such an endpoint is not required in this project.
12. For endpoints that require a request body, you only need to support JSON representations in the request body.
    - Requests to some endpoints, e.g., GET don't have a body. This point doesn't apply to such endpoints.
13. Any response bodies should be in JSON, including responses that contain an error message.
    - Responses from some endpoints, e.g., DELETE, don't have a body. This point doesn't apply to such endpoints.
    - In some cases error message may get generated by the web server and your code may not even get invoked. This point about JSON response body doesn't apply in such cases.
14. Any request to an endpoint that will send back a response with a body must allow 'application/json' in the Accept header. If a request doesn't have such a header, it should be rejected.

User Details

1. You must support the ability for users of the application to create user accounts. There is no requirement to edit or delete users.
2. You may choose from the following methods of handling user accounts
    - You can handle all account creation and authentication yourself.
    - You can use a 3rd party authentication service (e.g., Auth0 or Google).
3. You must provide a URL where a user can provide a username/email address, and a password to login or to create a user account. This can be a URL in your app that redirects to Google or Auth0.
4. You must have a User entity in your database which stores at least the unique user ID of each user of your application.
    - The first time someone logs in and generates a JWT in your app they must be added as a user in the User entity of your database.
5. Requests for the protected resources must use a JWT for authentication. So you must show the JWT to the user after the login. You must also show the user's unique ID after login.
6. The choice of what to use as the user's unique ID is up to you.
    - You can use the value of "sub" from the JWT as a user's unique ID. But this is not required.
7. You must provide an unprotected endpoint **GET /users** that returns all the users currently registered in the app, even if they don't currently have any

relationship with a non-user entity. The response does not need to be paginated.
    - Minimally this endpoint should display the unique ID for a user. Beyond that it is your choice what else is displayed.
8. There is no requirement for an integration at the UI level between the login page and the REST API endpoints.
    - The graders will login using the app's login/create user account URL.
    - Copy the JWT displayed and manually paste it in the Postman environment files to run the tests.

Status Codes

Your application should support at least the following status codes.

1. 200
2. 201
3. 204
4. 401
5. 403
6. 405
7. 406

Submission Details

You need to submit 4 files:

1. A file **YourONID_project.pdf** that should contain the following things
    - The URL where your application is deployed on GCP.
    - The URL for account creation/login (can be the same as above).
    - A data model section that includes the following information:
        - For all entities (user and non-user) their properties
            - For each property,
                - Its type
                - Whether or not it is required
                - Valid values
        - A description of the relationship between the non-user entities.
        - A description of how you are modeling the user entity in your application, including
            - What is the unique identifier for a user in your data model.

        - If a request needs to supply a user identifier, what needs to be specified by the person making the request.
        - How your application maps a supplied JWT to the identifier of a user.
        - What is the relationship between the user entity and non-user entity.
    - An API specification that details
        - All endpoints, i.e., the URL and the method.
        - For every endpoint, whether or not it is protected.
        - Status codes that an endpoint can return.
        - Sample requests and responses for these status codes.
    - You must use the API doc for Assignment 3 as a guide to what you need to specify.
2. A file **YourONID_project.postman_collection.json** with a Postman Collection of a test suite. See below for details about what tests this collection should contain.
3. A file **YourONID_project.postman_environment.json** with the Postman Environment your Postman test collection uses. This environment must contain at least the following 2 variables
    -
        - jwt1
        - jwt2
    - The tests for protected endpoints must use the value of one of the jwt variables as the JWT for authentication.
    - Additionally, if you have any endpoints that require a user_id, then your Postman Environment file must also contain the following variables
        - user_id1
        - user_id2
    - If an endpoint requires a user_id in the URL, then the tests for this endpoint must use the variable user_id1 or user_id2
4. A file **YourONID_project.zip** that should include all the source code for your project. **Don't include** node_modules or Python env in the zip file.

Postman Test Collection

1. It must demonstrate create, read, update and delete operations for all non-user entities.
2. It must demonstrate read operations for all collections of non-user entities.
3. It must demonstrate creating and deleting relationships between non-user entities.
4. There must be at least one test per required status code showing things working as intended.

5. It must demonstrate user accounts working as intended
    - Show that entities created by one user can be read, updated and deleted by that user.
    - Show that entities created by one user cannot be read, updated and deleted by another user.
    - Show that requests to protected resources are rejected if the JWT is invalid.
    - Show that requests to protected resources are rejected if the JWT is missing.
6. The tests must verify at least the response code. It is not required that the tests verify the response body. However, the response body must match your API spec.
7. In your spec you need to specify valid values for the properties in the request body. However, your application and your tests don't need to cover input validation. You can assume that the input will be valid per your specification.
8. A possible Postman test to show that one user can't read, edit or delete an entity created by another user is the following;
    - Call your REST API to create an entity using jwt1 and then have a test that shows that attempts to read, edit and delete that entity using jwt2 are rejected.
9. A possible Postman test to verify the endpoints that create and remove relationships is as follows
    - Show the resources before the relationship is created (or removed)
    - Create (or remove) the relationship
    - Show the resources after the relationship has been created (or removed)