E-Z Beach

Simplify Your Day at the Beach

Service Layers

Dustin Kent
07/16/2023

Overview

The backend service for my web application will be developed using Express Node.js. It will serve as a reliable and efficient RESTful API that the frontend can communicate with to perform operations. My service will be structured into 3 layers to ensure a well-organized and maintainable codebase.

1. **Route Layer**: The route layer defines the specific paths and endpoints that the frontend can access to retrieve or modify data. These routes serve as the entry points for incoming HTTP requests. They are responsible for mapping the requested URLs to the corresponding controller endpoints.
2. **Controller Layer**: Each route will be associated with a specific controller, which acts as the middleware to handle the incoming HTTP requests. The controller's primary role is to process the incoming requests, extract any necessary data, and invoke the appropriate service methods. It acts as a bridge between the routes and the underlying service layer.
3. **Service Layer**: The service layer contains the business logic and data manipulation operations. Each controller endpoint will call specific service methods, which will handle the data operations, interact with the MongoDB database and execute the required logic. The service layer ensures separation of concerns and encapsulates the database operations, allowing for modularity and testability.

I think I can further break my application down into the following 5 layers, but I think I am incorporating front end in it (where I believe this is just asking about the backend I).

1. **Presentation Layer**: This layer handles the user interface and user interactions. It includes components like HTML, CSS, and JavaScript, the presentation layer is responsible for rendering the user interface and capturing user input.
2. **Application Layer**: The application layer contains the business logic of my application. It handles the processing and coordination of data, as well as the implementation of business rules. This layer is responsible for validating user input, handling requests, and orchestrating the overall functionality of my application.
3. **Data Layer**: The data layer manages the storage and retrieval of data. It interacts with the database to perform some CRUD operations. Since I'm using MongoDB, I may use Mongoose to interact with the database (not completely sure yet).
4. **Service Layer**: The service layer acts as a go-between, between the application layer and the data layer. It encapsulates the business logic and provides reusable services that can be used across different parts of my application.
5. **Integration Layer**: Since my application will need to interact with external services/APIs, I will need an integration layer. This layer handles the communication with external systems, such as payment gateways, email services, or third-party APIs.

These layers work together to provide a structured and modular architecture for my backend. They promote code organization and maintainability. The specific implementation and complexity of each layer will vary and may change as my application is built and grows.

1. Get users:
    - Method: GET
    - URL: https://your-aws-domain/api/users
    - Purpose: Retrieve a list of users. This endpoint can be used to fetch user information for administrative purposes or display user profiles.
    - Example Requests:
        - Success Request:

curl --request GET --url https://your-aws-domain/api/users

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "userId": "123456",
    "name": "John Doe",
    "email": "johndoe@example.com",
    "phone": "123-456-7890",
    "address": {
      "street": "123 Main Street",
      "city": "City",
      "state": "State",
      "postalCode": "12345",
      "country": "Country"
    }
  },
  // Additional user objects
]

```

Error Response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
  "error": "Users not found"
}

```

Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

[Client (User Interface: (“Admin”) “Users Page”)] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [User Controller] -> [User Service] -> [User Repository (MongoDB)]

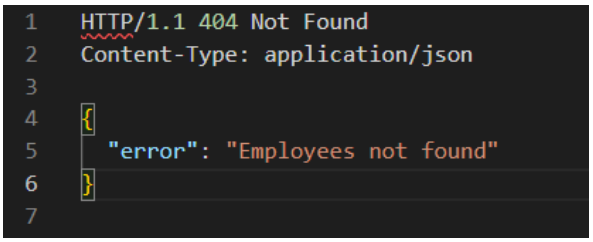
2. Get employees:
    - Method: GET
    - URL: https://your-aws-domain/api/employees
    - Purpose: Retrieve a list of employees. This endpoint can be used to fetch employee information for administrative purposes or display employee profiles.
    - Example Requests:
        - Success Request:

curl --request GET --url https://your-aws-domain/api/employees

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "employeeId": "987654",
    "name": "John Smith",
    "email": "johnsmith@example.com",
    "phone": "123-456-7890",
    "address": {
      "street": "123 Main Street",
      "city": "City",
      "state": "State",
      "postalCode": "12345",
      "country": "Country"
    }
  },
  // Additional employee objects
]
```

Error Response:



```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
  "error": "Employees not found"
}
```

Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

[Client (Admin Employees Page)] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [Employee Controller (Get Employees)] -> [Employee Service] -> [Employee Repository (MongoDB)]

3. Get locations:
    - Method: GET
    - URL: https://your-aws-domain/api/locations
    - Purpose: Retrieve a list of beach locations. This endpoint can be used to fetch location information for displaying available locations or managing location data.
        - Example Requests:
            - Success Request:

curl --request GET --url https://your-aws-domain/api/locations

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "locationId": "123456",
    "name": "Beach A",
    "address": "123 Beach Street",
    "coordinates": {
      "latitude": 25.123456,
      "longitude": 115.25698
    }
  },
  // Additional location objects
]
```

Error Response:

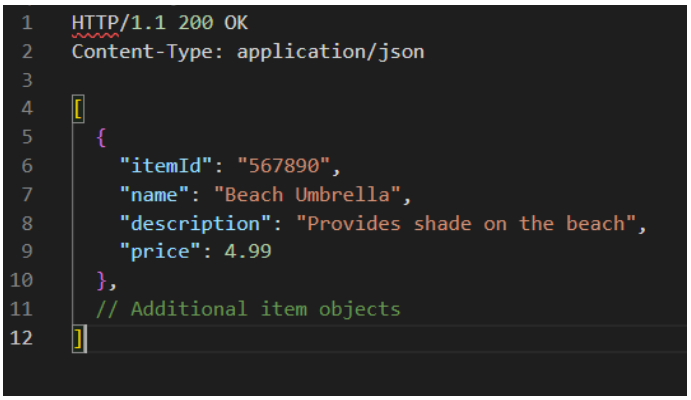
```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
  "error": "Locations not found"
}
```

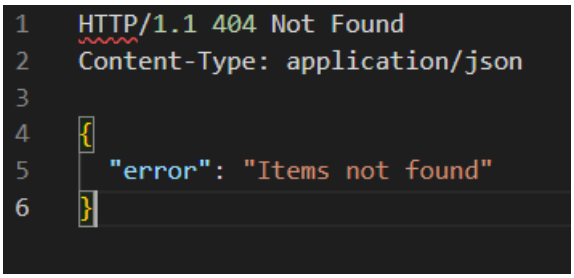
Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

4. Get items:
    - Method: GET
    - URL: https://your-aws-domain/api/items
    - Purpose: Retrieve a list of beach items. This endpoint can be used to fetch item information for displaying available items or managing item data.
    - Example Requests:
        - Success Request:

curl --request GET --url https://your-aws-domain/api/items



Error Response:



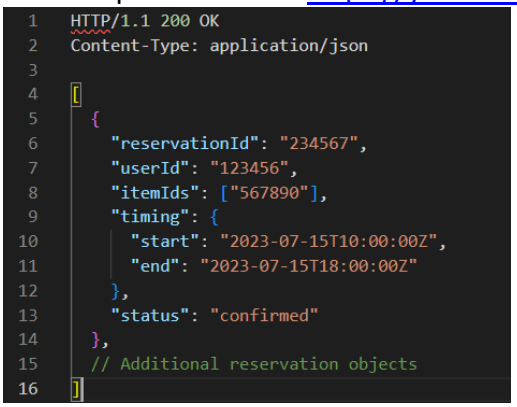
Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

[Client (Schedule a Pickup Page)] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [Item Controller (Get Items)] -> [Item Service] -> [Item Repository (MongoDB)]

5. Get reservations:
    - Method: GET
    - URL: https://your-aws-domain/api/reservations
    - Purpose: Retrieve a list of reservations. This endpoint can be used to fetch reservation information for administrative purposes, display user reservations, or manage reservation data.
    - Example Requests:
        - Success Request:

curl --request GET --url https://your-aws-domain/api/reservations



Error Response:

Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

[Client (Employee “View Jobs” / User “History Page” / Admin “View Reservations”)] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [Reservation Controller (Get Reservations)] -> [Reservation Service] -> [Reservation Repository (MongoDB)]

6. Get jobs:

- Method: GET
- URL: https://your-aws-domain/api/jobs
- Purpose: Retrieve a list of jobs. This endpoint can be used to fetch job information for administrative purposes, display job details, or manage job data.
- Example Requests:
    - Success Request:

curl --request GET --url https://your-aws-domain/api/jobs

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "jobId": "123",
    "locationId": "12345",
    "employeeId": "123456",
    "itemIds": ["123456"],
    "timing": {
      "start": "2023-07-16T09:00:00Z",
      "end": "2023-07-16T17:00:00Z"
    },
    "status": "accepted"
  },
  // Additional job objects
]
```

Error Response:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
  "error": "Jobs not found"
}
```

Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

[Client (Employee “View Jobs” / User “History Page” / Admin “View Reservations”)] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [Reservation Controller (Get Reservations)] -> [Reservation Service] -> [Reservation Repository (MongoDB)]

7. Create User:

- Method: POST
- URL: https://your-aws-domain/api/users
- Purpose: This endpoint is used to create a new user in the system. It receives the user's name, email, and password as input and generates a unique identifier for the user. The created user is then stored in the database, allowing them to access the application with their credentials.
- Example requests:

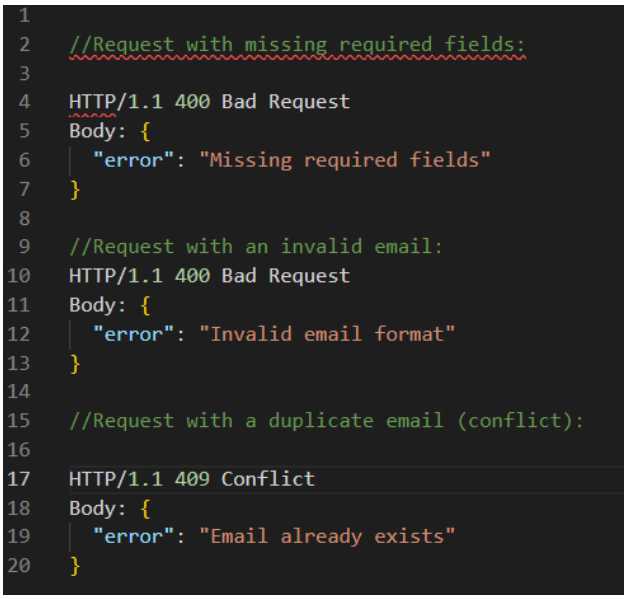
curl --request POST --url https://api.example.com/users \

--header 'Content-Type: application/json' \
--data '{
  "name": "John Doe",
  "email": "johndoe@example.com",
  "password": "secretpassword"
}'

Success Response:

```
HTTP/1.1 201 Created
Body: {
  "id": "123456789",
  "name": "John Doe",
  "email": "johndoe@example.com"
}
```

Error Responses:



```
//Request with missing required fields:

HTTP/1.1 400 Bad Request
Body: {
  "error": "Missing required fields"
}

//Request with an invalid email:
HTTP/1.1 400 Bad Request
Body: {
  "error": "Invalid email format"
}

//Request with a duplicate email (conflict):

HTTP/1.1 409 Conflict
Body: {
  "error": "Email already exists"
}
```

Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

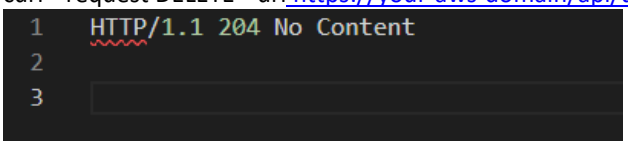
Diagram/Route:

[Client (User Interface: “Account Page”)] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [User Controller (Create User)] -> [User Service] -> [User Repository (MongoDB)]

8. Delete User
   - Method: DELETE
   - URL: https://your-aws-domain/api//users/{userId} Purpose: This endpoint is used to delete a user from the system. It receives the user's ID as input and removes the corresponding user from the database. Example requests:

- Request

curl --request DELETE --url https://your-aws-domain/api/users/123456789



```
HTTP/1.1 204 No Content


```

Error Response:

```
HTTP/1.1 404 Not Found
Body: {
  "error": "User not found"
}
```

Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

[Client (User Interface: “Account Page”)] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [User Controller] -> [User Service] -> [User Repository (MongoDB)]

9. Create Job/Reservation:

    - Method: POST
    - URL: https://your-aws-domain/api/jobs
    - Purpose: This endpoint is used to create a new job/reservation in the system. It receives the necessary details such as the location, employee, items, and timing as input. The job/reservation is then stored in the database, allowing users and employees to view and manage the created job.

Example requests::

curl --request POST --url https:// https://your-aws-domain/api jobs \
--header 'Content-Type: application/json' \
--data

```
'{
    "locationId": "12345",
    "employeeId": "67890",
    "itemIds": ["item1", "item2"],
    "timing": {
      "start": "2023-07-16T09:00:00Z",
      "end": "2023-07-16T17:00:00Z"
    }
  }'
```

Success:

```
HTTP/1.1 201
{
    "message": "Job created successfully",
    "jobId": "ABCDE"
  }
```

Error Response:

```
(HTTP/1.1 400 Bad Request)
{
  "error": "Invalid input. Please provide valid location, employee, items, and timing details."
}

(HTTP/1.1 404 Not Found)
{
  "error": "Location or employee not found. Please provide valid IDs."
}
```

Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

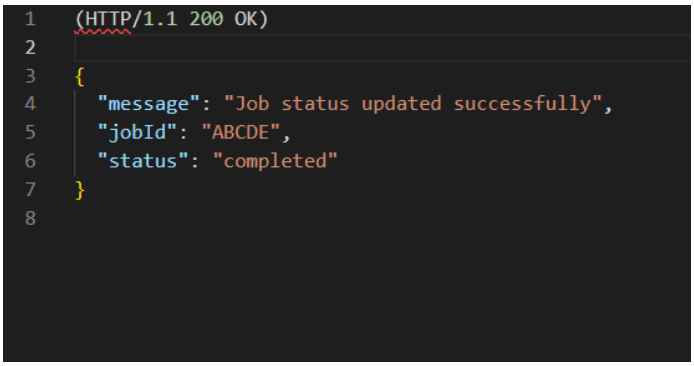
[Client (User Interface: "Schedule a Pickup Page")] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [Job Controller (Create Job)] -> [Job Service] -> [Job Repository (MongoDB)]

10. Accept/Complete Job:

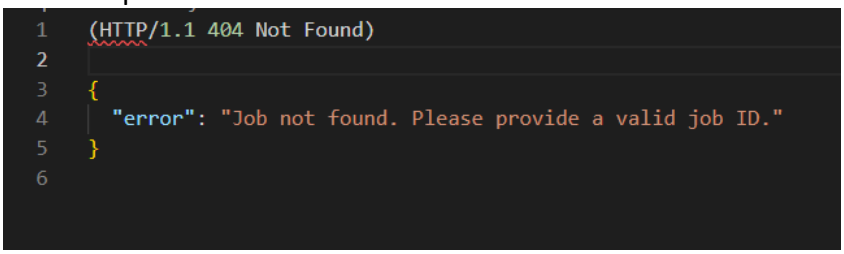
- Method: PUT
- URL: https://your-aws-domain/api/jobs/{jobId}/status
- Purpose: This endpoint is used by the employee to accept or complete a job. The employee provides the job ID as a path parameter and updates the status field accordingly. Accepting a job changes the status to "accepted," while completing a job changes the status to "completed." This endpoint allows for tracking the progress of the job and notifying other users or systems about its status.
- Example requests:

```
curl --request PUT --url https://your-aws-domain/api/jobs/ABCDE/status \
--header 'Content-Type: application/json' \
--data '{
  "status": "accepted"
}'
curl --request PUT --url https://your-aws-domain/api /jobs/ABCDE/status \
--header 'Content-Type: application/json' \
--data '{
  "status": "completed"
}'
```

Success Response:



Error Response:



Error Response: (for demonstration purposes only, the actual endpoint won't throw an error as long as the server is running)

Diagram/Route:

[Client (Employee User Interface: "View Jobs Page")] -> [API Gateway] -> [Load Balancer] -> [Application Server (Node.js)] -> [Job Controller (Accept/Complete Job)] -> [Job Service] -> [Job Repository (MongoDB)]

I am still trying to work the database and routes out completely. I feel with my application, the same basic data will be stored in the database with users and employees having access to the same basic information as far as jobs/reservations pertaining to their own account and the admin will have access to everything, with access controls being implemented to only show what is required to the “user” or “employee” based on needs of access.

My endpoints at this time are not final as I will be making adjustments on them as I continue to work on this project.