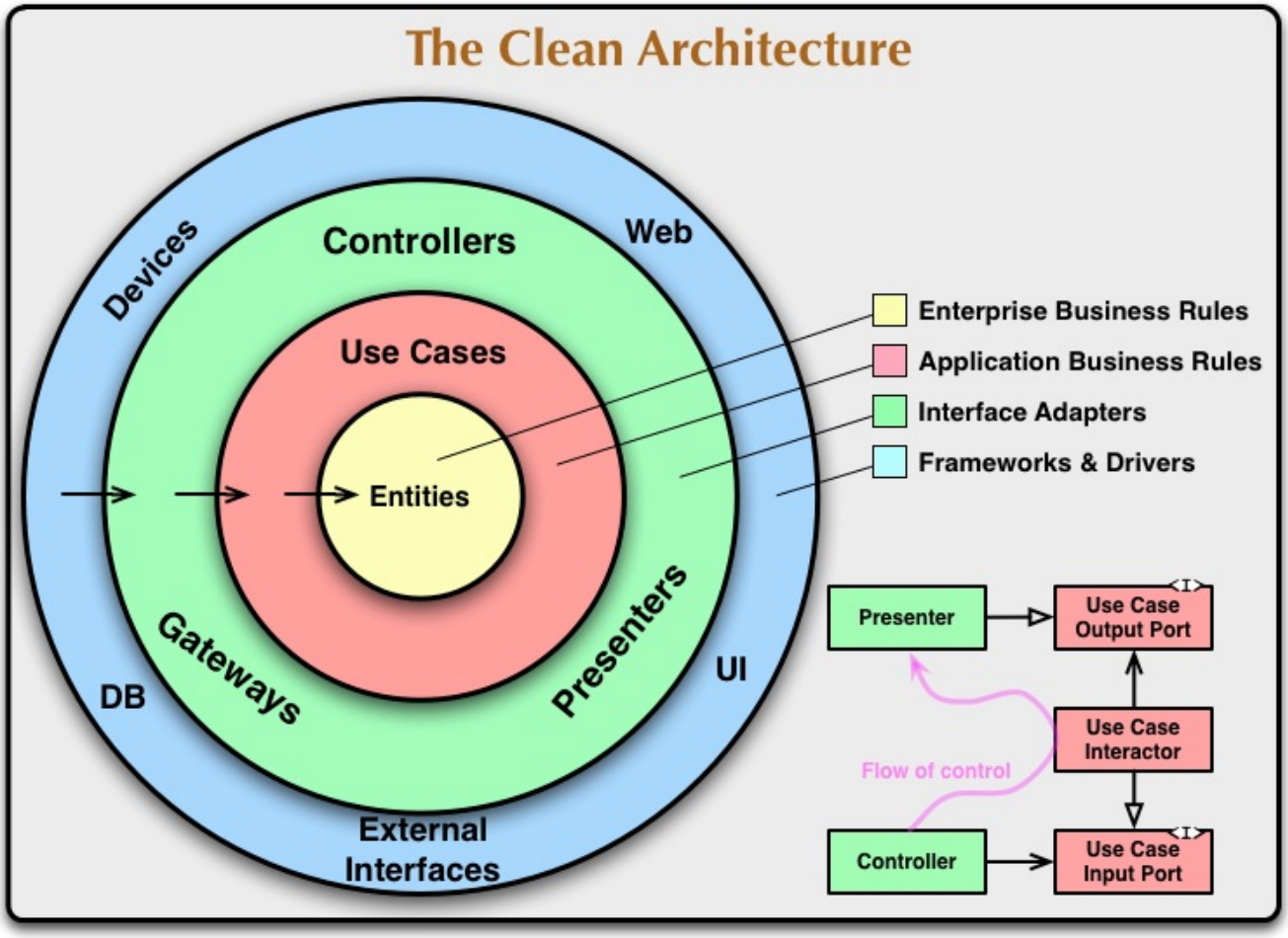
CS3219 Task B: CRUD Application Task

Name: Koh Vinleon
Matric Number: A0202155W
GitHub Link: https://github.com/glatiuden/CS3219-OTOT-TaskB

Task B1: Implementing Backend

This is an attempt in building a (semi) Clean Architecture Node.js backend.

Clean Architecture



Read more at Clean Coder Blog

Layer description:

- Entities: Contain enterprise business model/object
- Use Cases: Contain application business rules/logic
- Interface Adapter: Contains a set of adapters that convert data from entities/use-case layer to external dependencies such as DB or Web/HTTP
- Frameworks/Driver: Compose of frameworks and tools (DB, Web Frameworks)

Set Up

Database Used: Atlas MongoDB
Libraries Used: Winston, Nodemon, Mongoose, Lodash and Validatorjs
Please ensure you are in the `/backend` folder (`cd backend`).

Please create a `.env` file in the backend directory with the following credentials.

```
MONGO_USERNAME="admin"
MONGO_PASSWORD="3YHYkUdqNUMykugo"
MONGO_DB="cs3219-otot-task-b"
```

Install the necessary modules

```
npm install
```

Start the server

```
npm run dev
```

Design

- All the endpoints are structured in this format `{URL}/api/{COLLECTION_NAME}`.

Note API

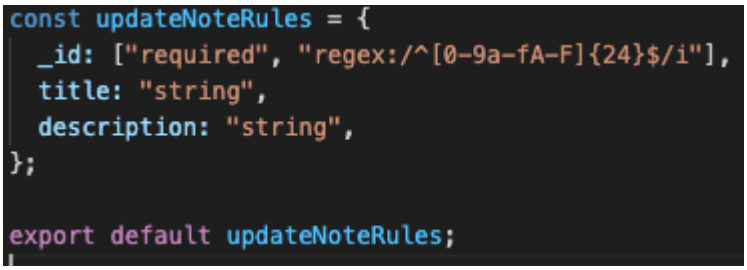
Method	Route	Description
POST	/api/note	Create a new note
GET	/api/note	Get all notes
GET	/api/note/:note_id	Get note by ID
PUT	/api/note	Update a note
DELETE	/api/note/:note_id	Soft delete a note
DELETE	/api/note/hard-delete/:note_id	Hard delete a note

- The results returned by the API must be `data`.
- For `GET`, there are two variants: one will get a specific record by `ID` while the other will get all the records from the database.
- For `DELETE`, there are two variants: one will perform a soft delete while the another will perform a hard delete.

Error Resiliency

API endpoints which requires route parameter (e.g. GET `/api/note/:note_id`) or message body (e.g. POST `/api/note/`) uses a validator middleware (Validatorjs) to ensure the required data is passed in along with the request. We can also specify the type of data or regex to be checked against with.

Example of a validator (update-note.ts)



```
const updateNoteRules = {
  _id: ["required", "regex:/^[0-9a-fA-F]{24}$/i"],
  title: "string",
  description: "string",
};

export default updateNoteRules;
```

- `_id` is a required data and the regex specified that it should be an `ObjectId`.
- `title` and `description` is expected to be a string.

If the data passed in fails the validation (e.g. missing route parameter, missing data in the message body, invalid data type, fails the regex), the response (error) code is `422`.

If there is an error encountered during the execution of a query, such as a record not found or an internal error, the response (error) code will be `404`.

Please refer to the demonstration section for the actual use cases.

Endpoint

- Localhost: http://localhost:5000
- Deployed Endpoint: https://asia-southeast1-cs3219-otot-task-b-325509.cloudfunctions.net/cs3219-otot-task-b-dev-app

▶ Run in Postman

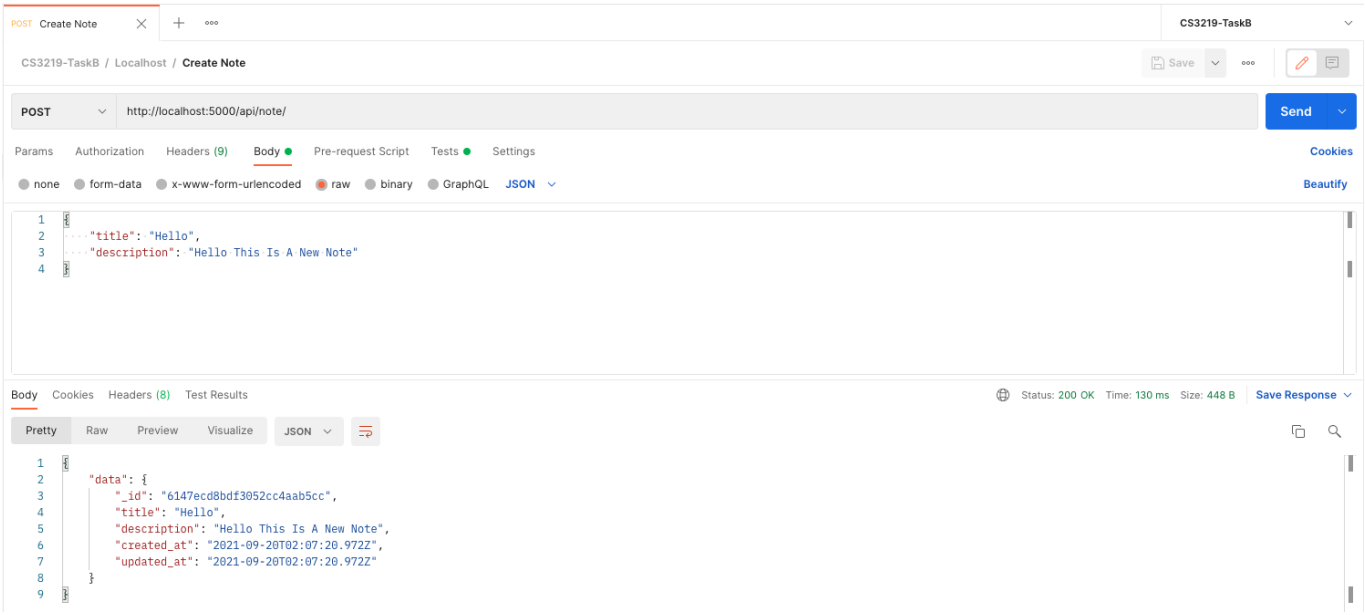
Alternatively, you may want to import it to your workspace via the JSON link or download the Postman JSON file in the Github Directory.

Demonstration

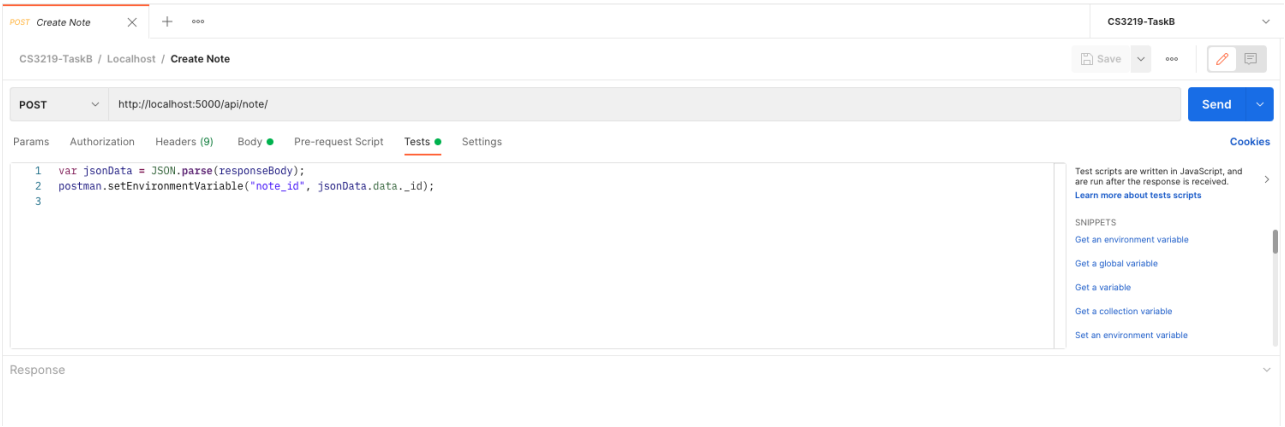
POST (CREATE)

- Method: `POST`
- Route: `/api/note`
- Description: Create new note
- Data (JSON): `title` (required), `description` (required)

Success (200)

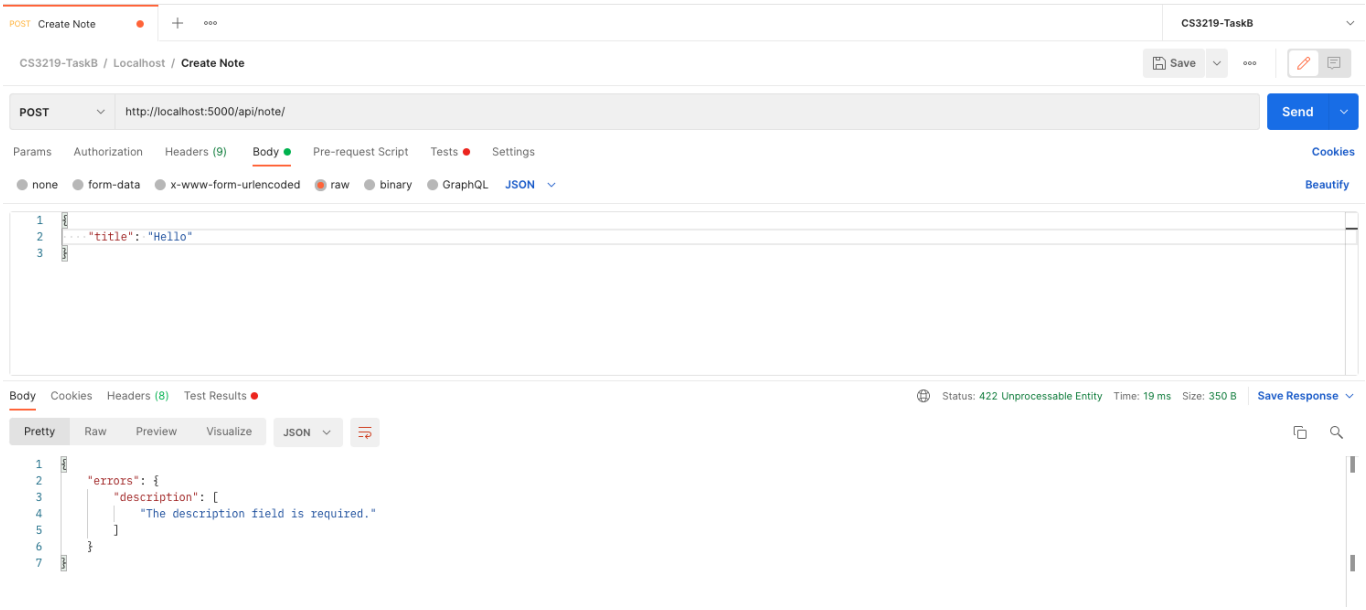


- For ease of demonstration and testing, the `note_id` returned in the body will be saved as a variable in Postman's local environment to be used in the subsequent requests.

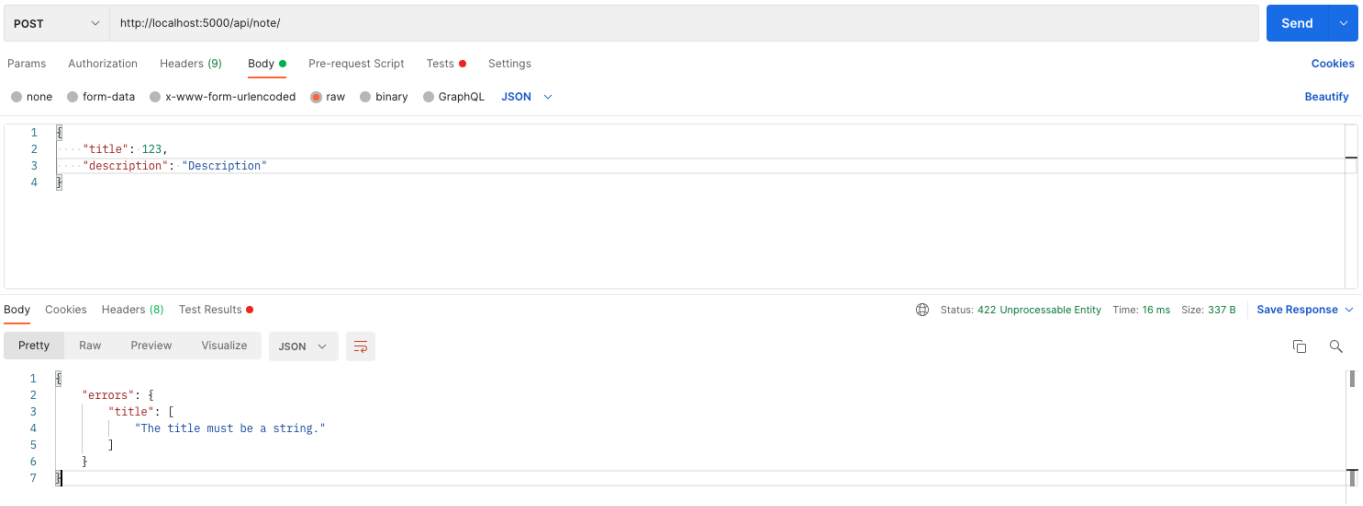


Error (422)

- Occurs due to missing required data fields (e.g. `title` and `description`).



- Occurs due to invalid data fields (e.g. `title` must be a string but was given an integer)

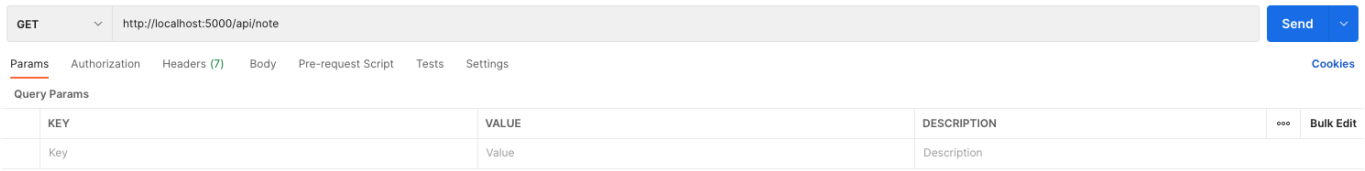


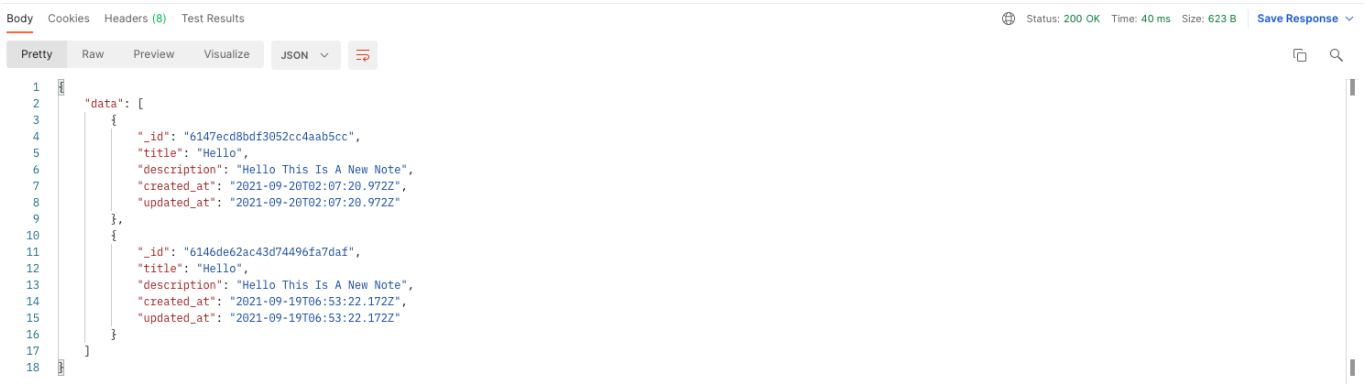
- Optimally, there can be an additional Error `404` if a note with the same `title` and `description` already exists in the database. However, this error is omitted from the implementation as it does not fit a note application's context as well as for the ease of testing.

GET (Retrieve)

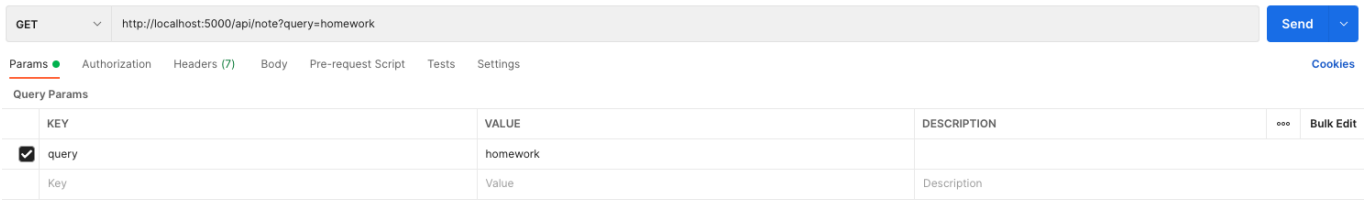
- Method: `GET`
- Route: `/api/note`
- Description: Get all notes
- Parameter: `query` (optional)

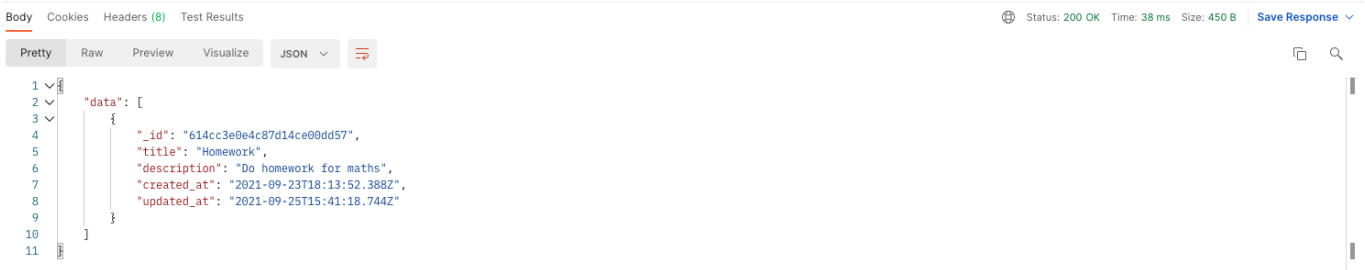
Success (200)





- To perform server side filtering, we can pass additional URL query parameters to search for notes with matching `title` or `description`. e.g. `/api/note?query=homework`.



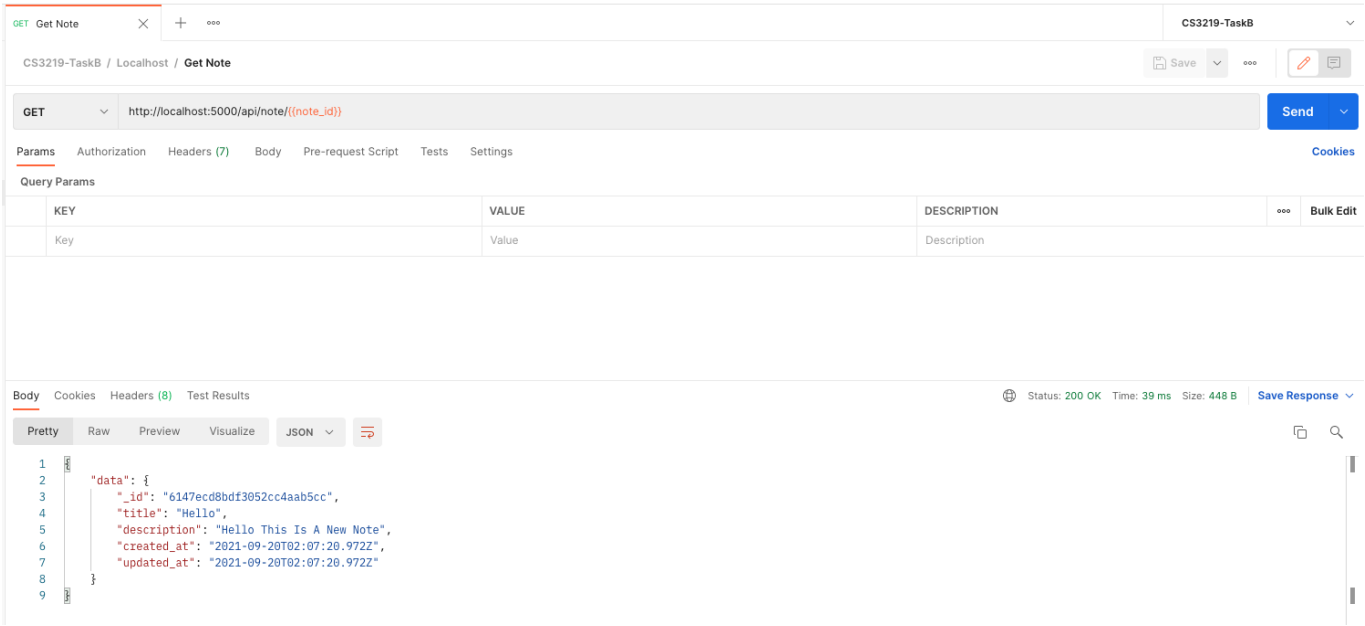


- Optionally, it can be an additional Error `204` (no content) if no notes are in the collections.
- I believe it's a debate between 204 and returning 200 with an empty array. For this task, I have chosen to follow 200 with an empty array.

GET (Retrieve By ID)

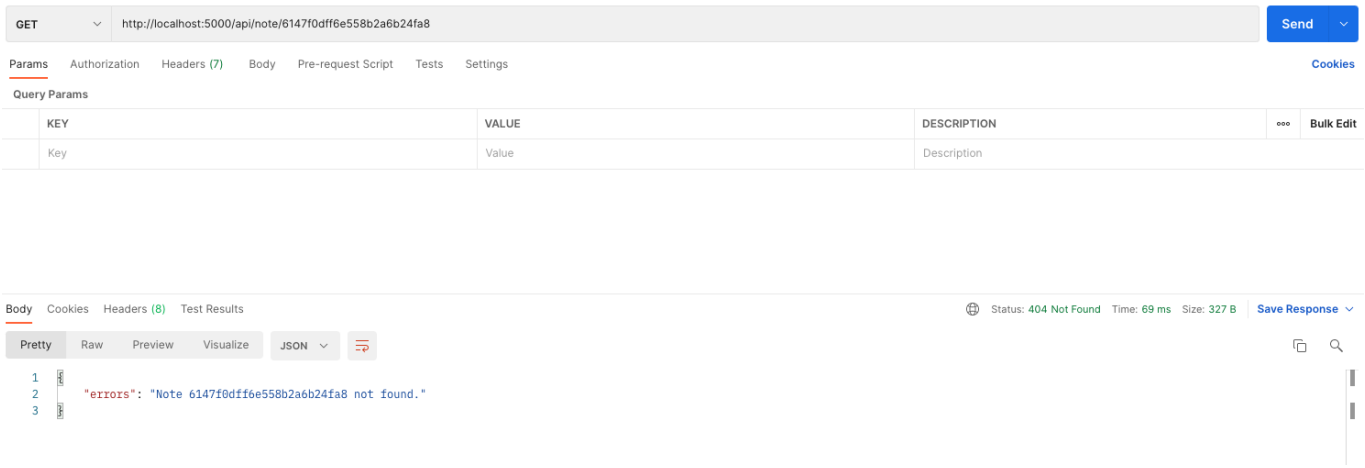
- Method: `GET`
- Route: `/api/note/:note_id`
- Description: Get note by ID

Success (200)



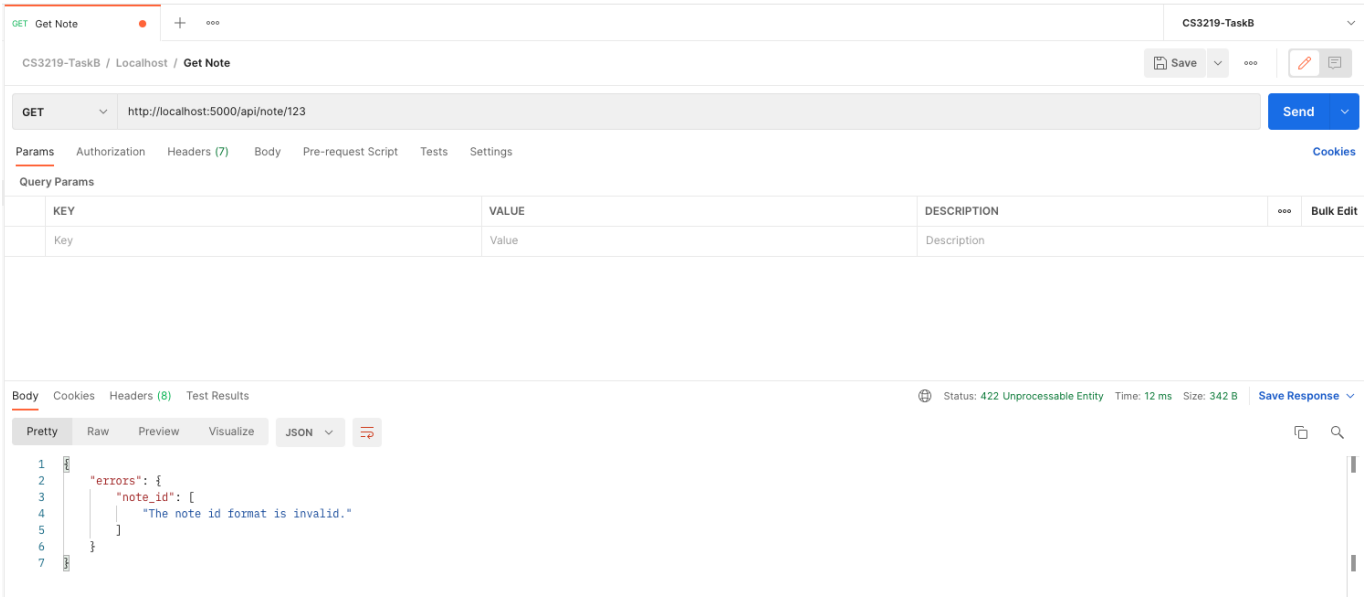
Error (404)

- Occurs when the parameter (`note_id`) is valid and present, but the record is not found in the database



Error (422)

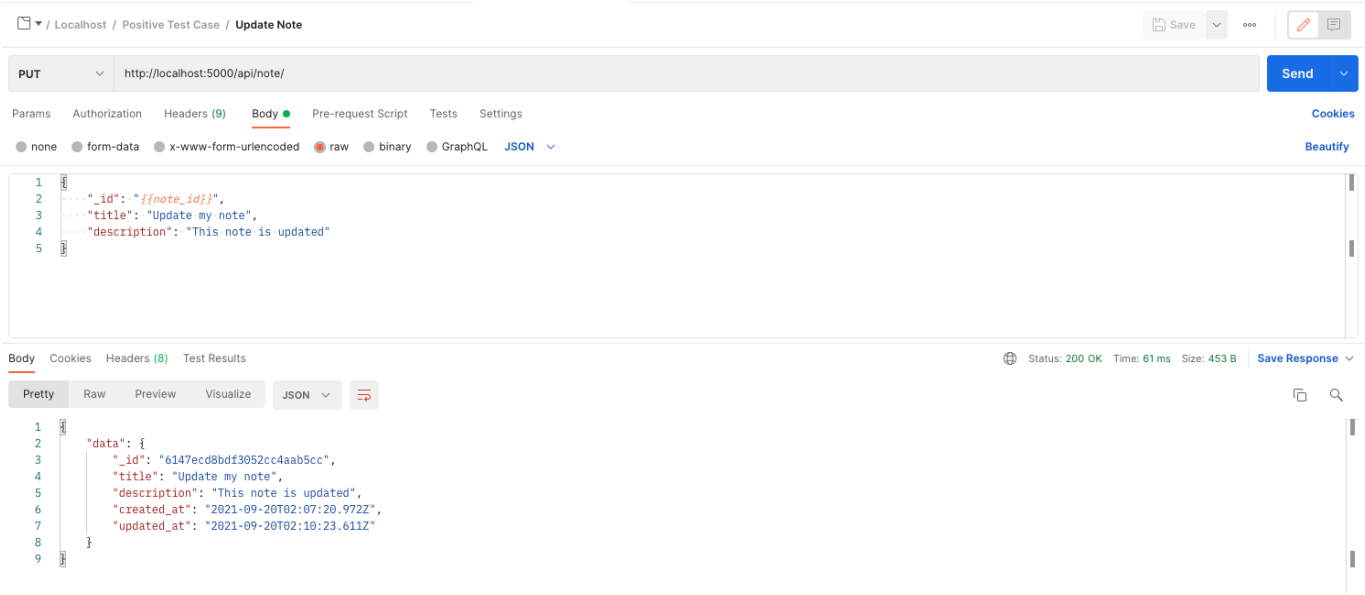
- Occurs when the parameter is invalid (e.g. the `note_id` is not a valid ObjectId).



PUT (Update)

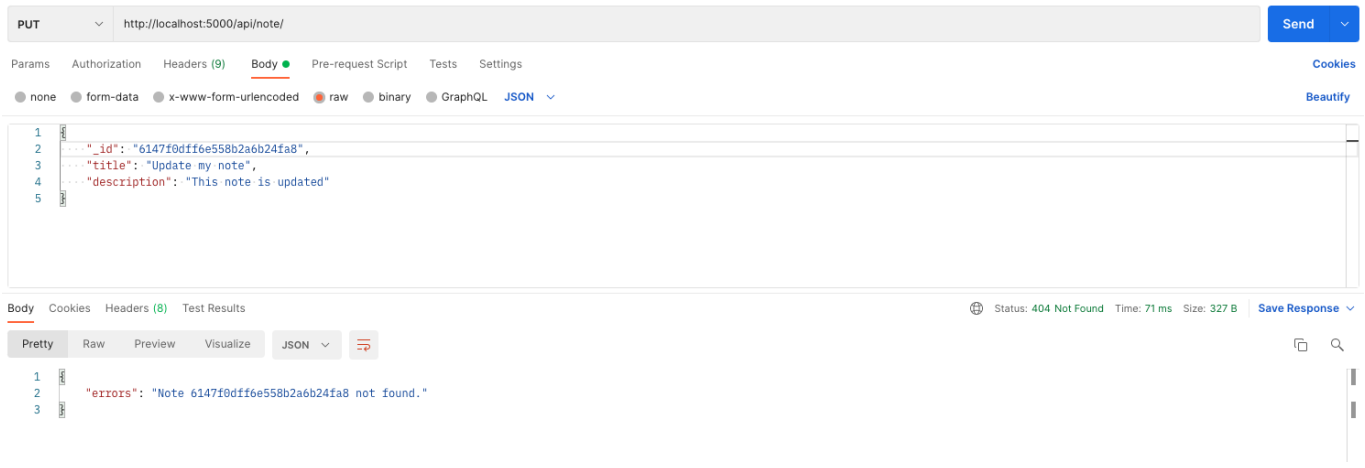
- Method: PUT
- Route: `/api/note/`
- Description: (Partial) Update existing note
- Data (JSON): `_id` (required), `title` (optional), `description` (optional)

Success (200)



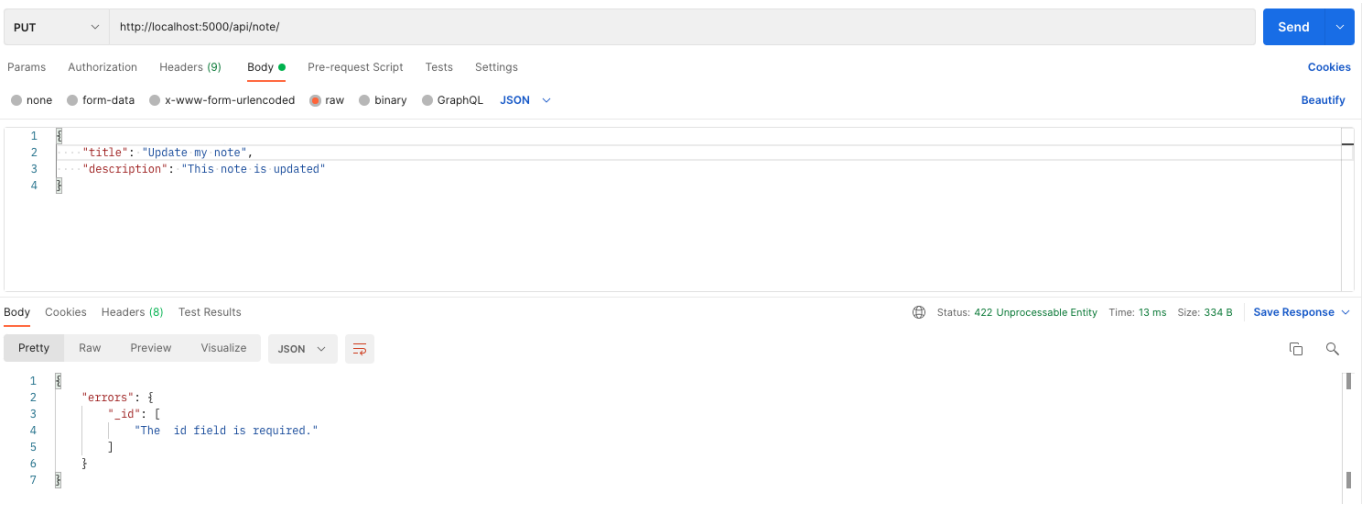
Error (404)

- Occurs when the parameter (`note_id`) is valid and present, but the record is not found in the database

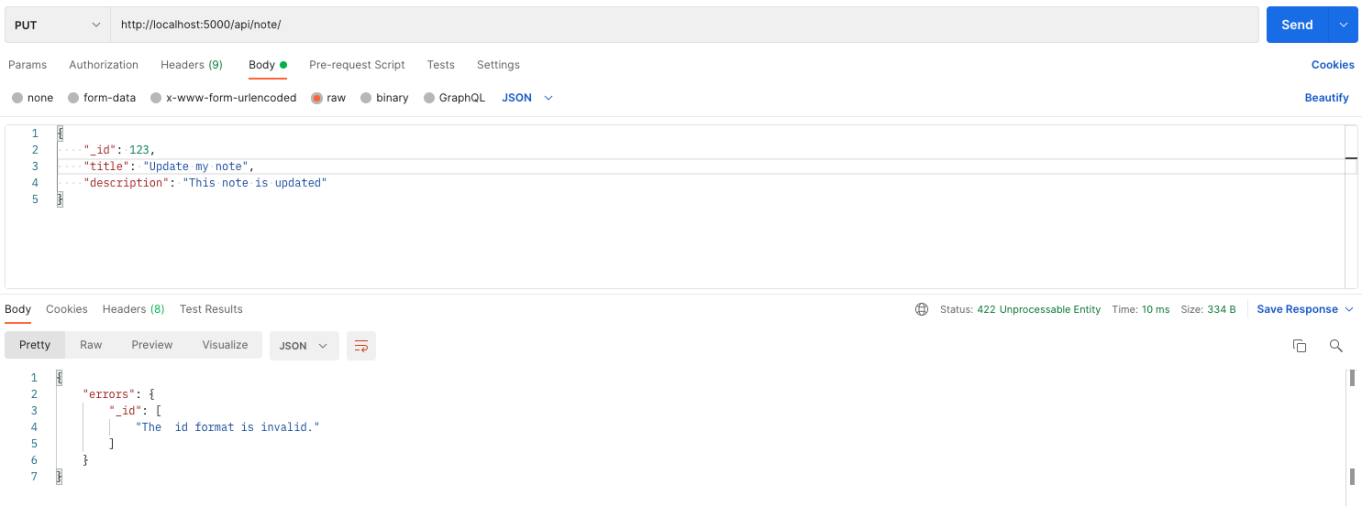


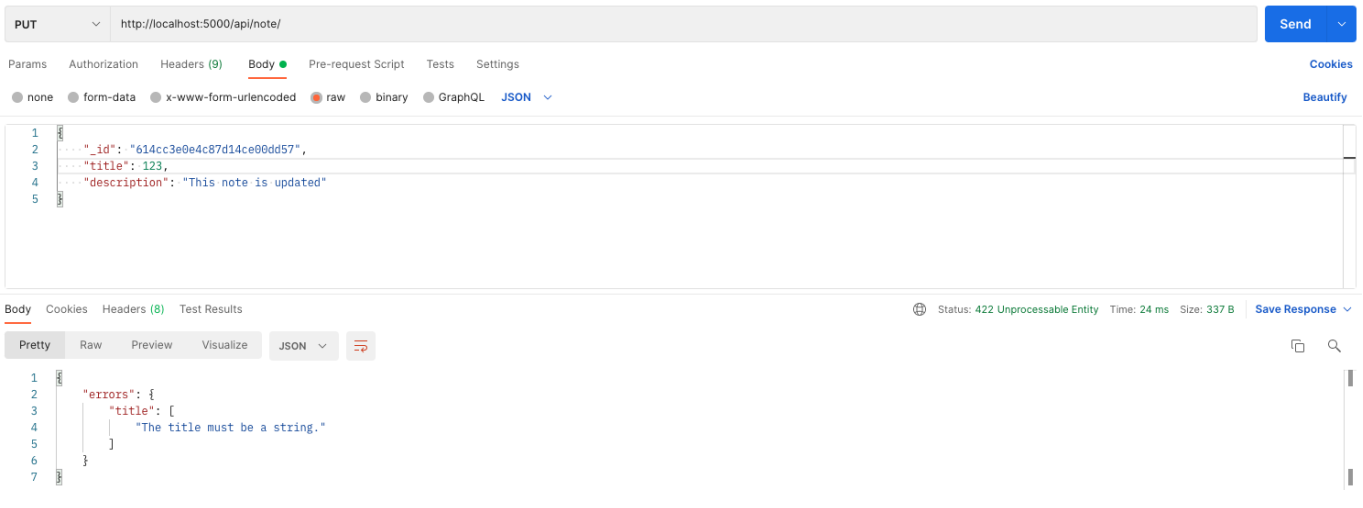
Error (422)

- Occurs when `_id` is missing
- As we are updating a note, `_id` is required to know which record to update.



- Occurs when `_id` is invalid (needs to be a valid ObjectID) or other fields (e.g. `title` needs to be a string)

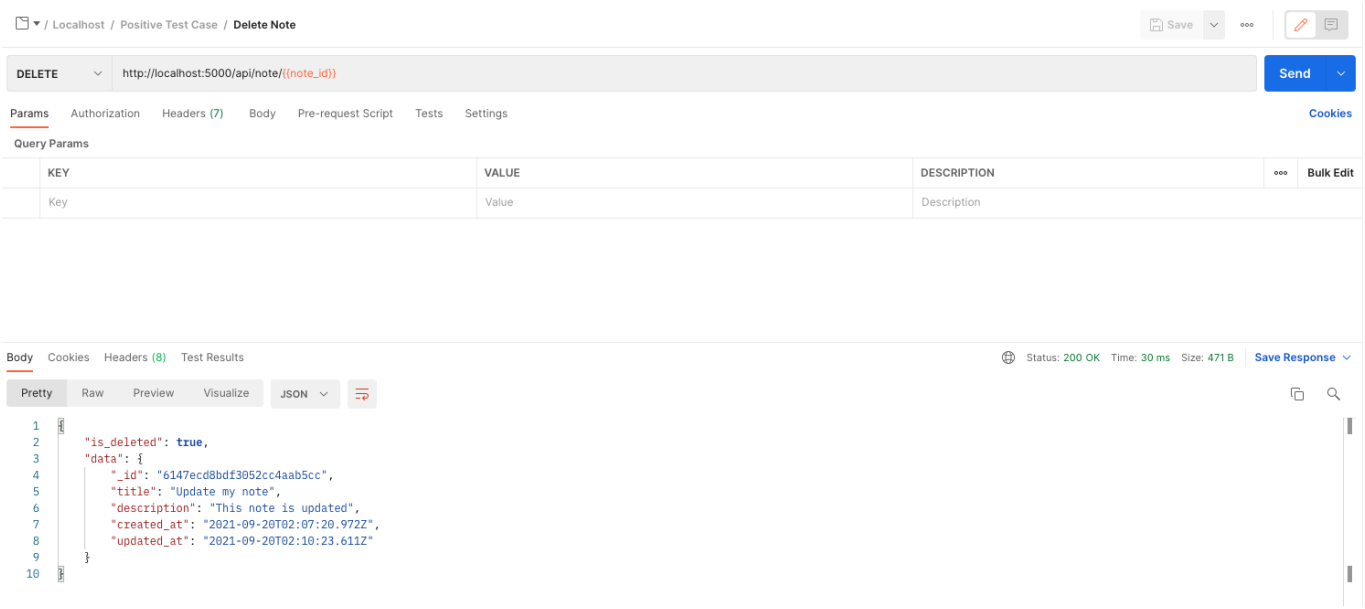




Delete (Soft Delete)

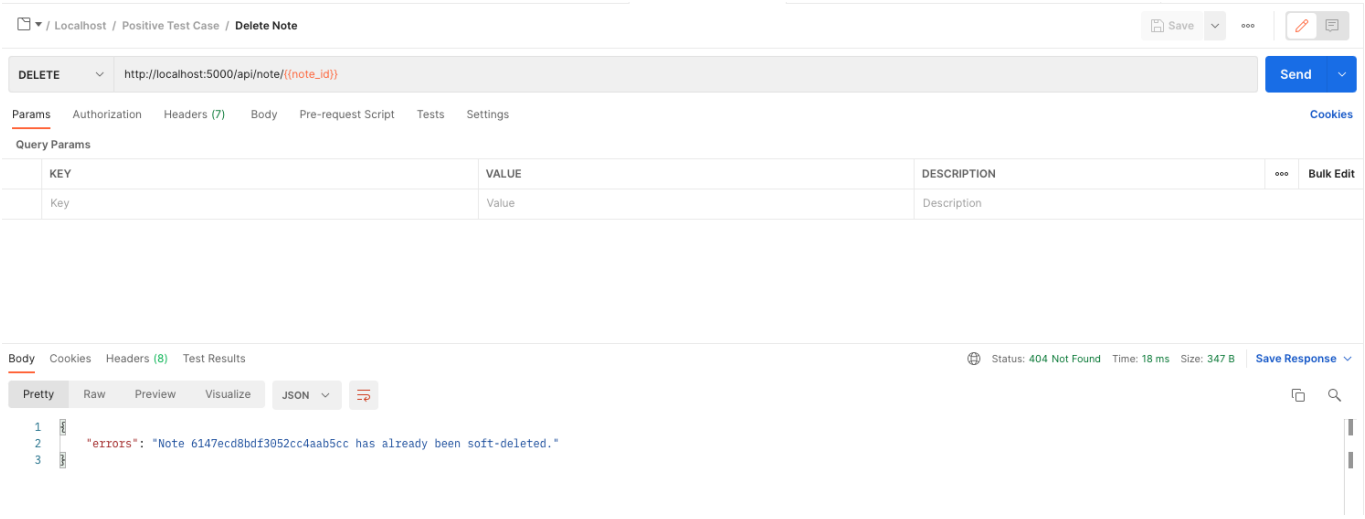
- Method: `DELETE`
- Route: `/api/note/:note_id`
- Description: Soft delete an existing note

Success (200)

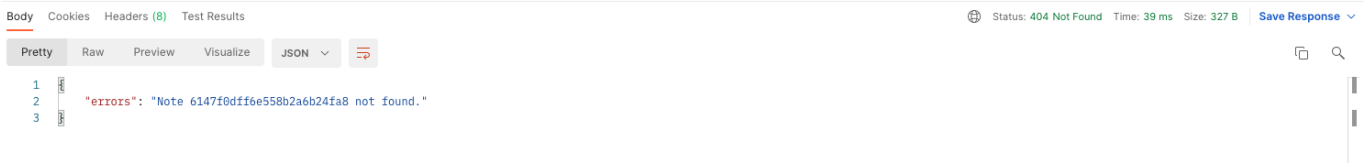


Error (404)

- Occurs when the parameter (`note_id`) is valid and present, but the record has already been soft-deleted

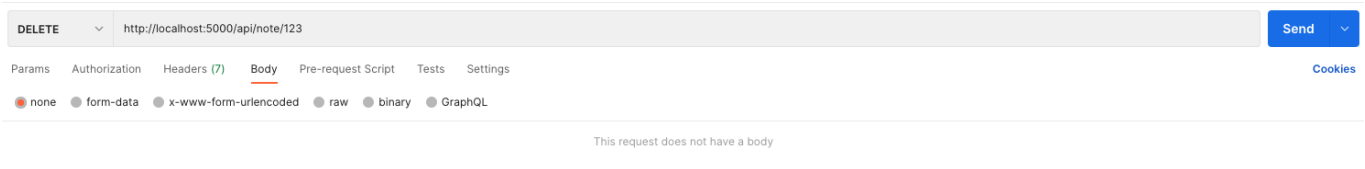


- Occurs when the parameter (`note_id`) is valid and present, but the record is not found in the database



Error (422)

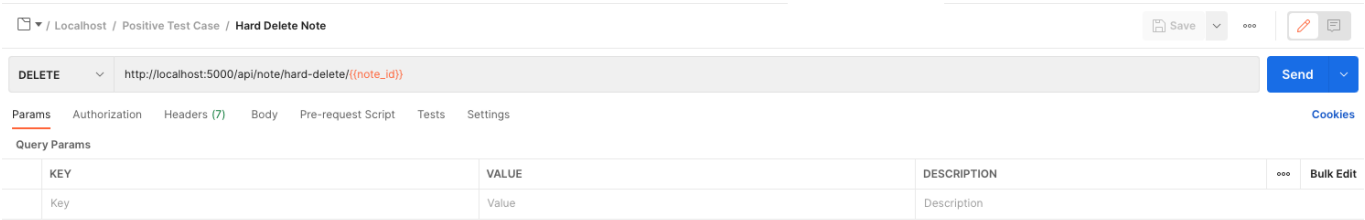
- Caused by an invalid parameter (In this scenario, the `note_id` is not a valid ObjectId).
- As we are soft deleting a note, `_id` is required to know which record to soft delete.

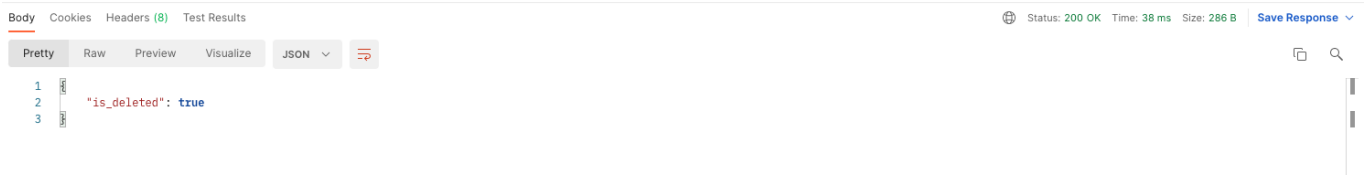


Delete (Hard Delete)

- Method: `DELETE`
- Route: `/api/note/hard-delete/:note_id`
- Description: Hard delete an existing note

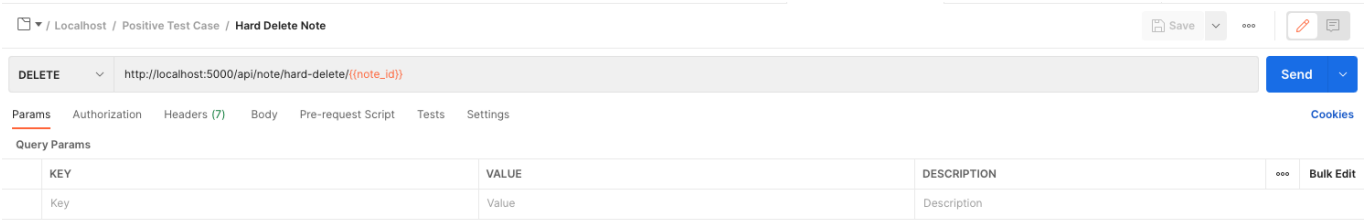
Success (200)

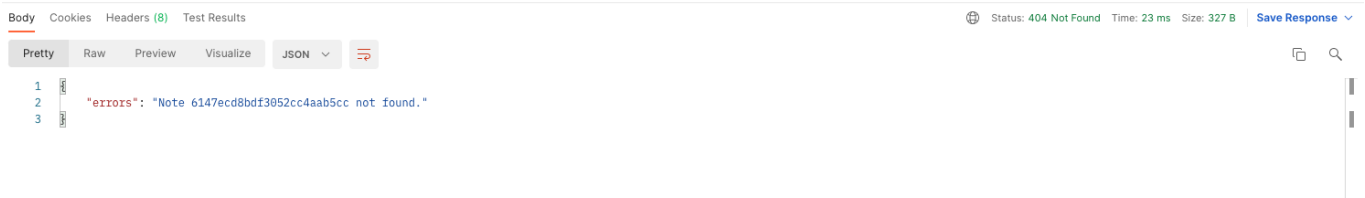




Error (404)

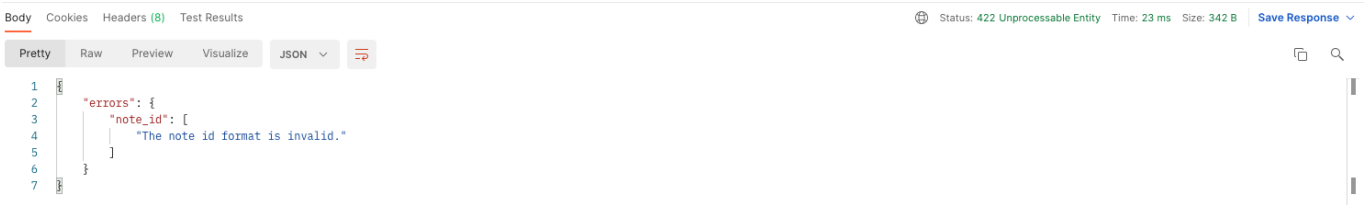
- Occurs when the parameter (`note_id`) is valid and present, but the record is not found in the database





Error (422)

- Caused by an invalid parameter (In this scenario, the `note_id` is not a valid ObjectId).
- As we are hard deleting a note, `_id` is required to know which record to hard delete.



References

- Using Clean Architecture for Microservice APIs in Node.js with MongoDB and Express
- Rules for clean code
- Node clean code architecture
- Application layer - use-cases
- Domain-driven Design articles
- Screaming architecture
- What is screaming architecture
- Clean architecture use-case structure
- Denormalize data
- Mongoose Database
- Expression documentation on API
- Bodyparser
- Winston-express for HTTP logging
- Winston for error logging
- Regex route express

Task B2: Testing through Continuous Integration (CI)

Test Frameworks: Mocha & Chai

Set Up

The tests are split into positive and negative test cases, which will test all the available methods, which consist of `POST` (create), `GET` (retrieve by ID & retrieve all), `PUT` (update) and `DELETE` (soft delete and hard delete). This ensures that the API endpoints status and responses are accurate.

- To test locally, we can run the test via `npm run test` command.

Postive Test Cases

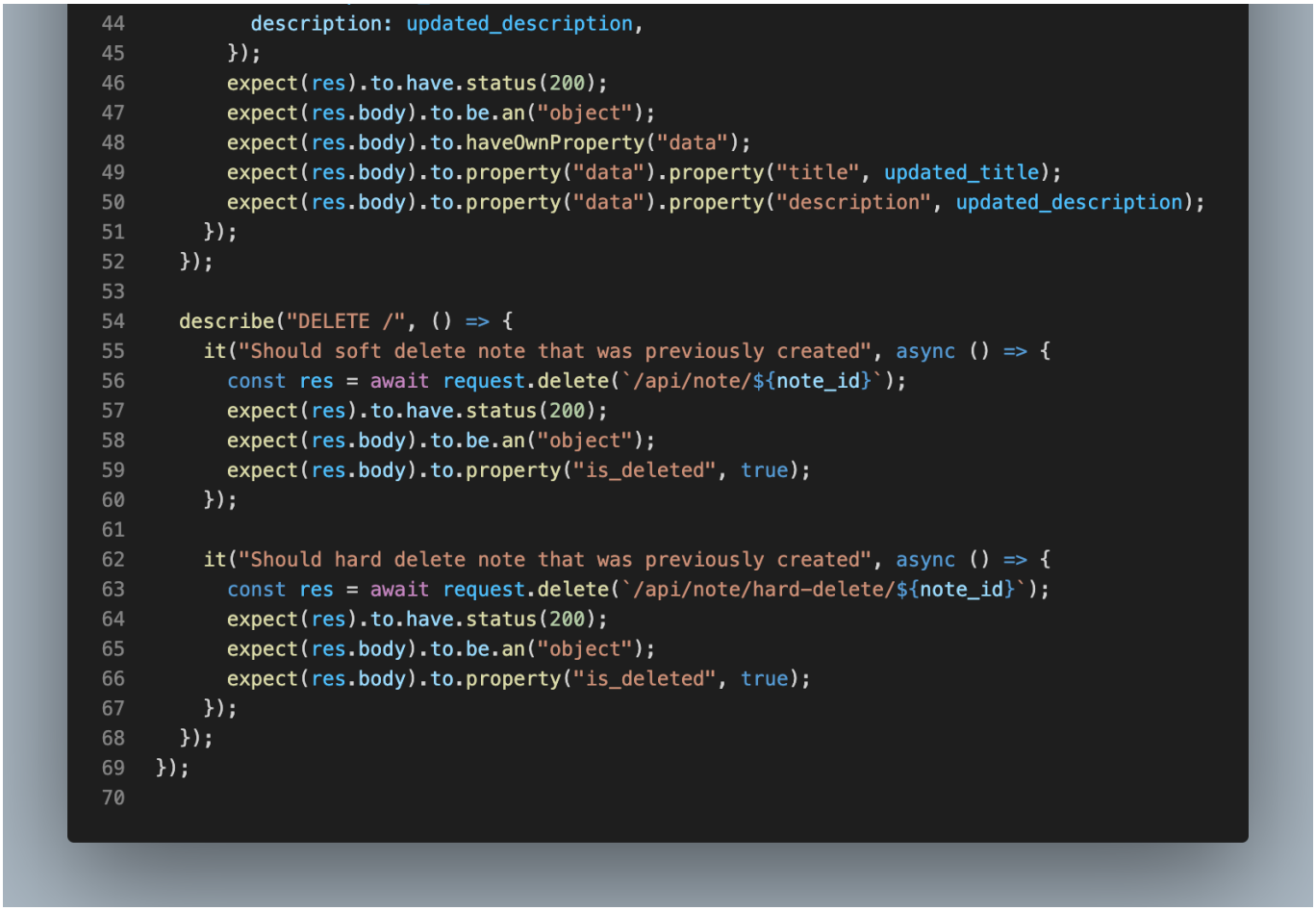
```
describe("Notes Postive Test Cases", () => {
  let note_id = ""; // Used in retrieval & deletion test
  const title = "New note from test";
  const description = "Note created using test";
  const updated_title = "Updated note from test";
  const updated_description = "This note is updated in test";

  describe("POST /", () => {
    it("Should create a note", async () => {
      const res = await request.post("/api/note").send({ title, description });
      expect(res).to.have.status(200);
      expect(res.body).to.be.an("object");
      expect(res.body).to.haveOwnProperty("data");
      expect(res.body).to.property("data").property("title", title);
      expect(res.body).to.property("data").property("description", description);
      note_id = _.get(res.body, "data._id");
    });
  });

  describe("GET /", () => {
    it("Should fetch all notes", async () => {
      const res = await request.get("/api/note/");
      expect(res).to.have.status(200);
      expect(res.body).to.be.an("object");
      expect(res.body).to.haveOwnProperty("data");
      expect(_.get(res.body, "data")).to.be.an("array");
    });

    it("Should fetch single note that was previously created", async () => {
      const res = await request.get(`/api/note/${note_id}`);
      expect(res).to.have.status(200);
      expect(res.body).to.be.an("object");
      expect(res.body).to.haveOwnProperty("data");
      expect(res.body).to.property("data").property("title", title);
      expect(res.body).to.property("data").property("description", description);
    });
  });

  describe("PUT /", () => {
    it("Should update note that was previously created", async () => {
      const res = await request.put("/api/note").send({
        _id: note_id,
        title: updated_title,
```



```
        description: updated_description,
      });
      expect(res).to.have.status(200);
      expect(res.body).to.be.an("object");
      expect(res.body).to.haveOwnProperty("data");
      expect(res.body).to.property("data").property("title", updated_title);
      expect(res.body).to.property("data").property("description", updated_description);
    });
  });

  describe("DELETE /", () => {
    it("Should soft delete note that was previously created", async () => {
      const res = await request.delete(`/api/note/${note_id}`);
      expect(res).to.have.status(200);
      expect(res.body).to.be.an("object");
      expect(res.body).to.property("is_deleted", true);
    });

    it("Should hard delete note that was previously created", async () => {
      const res = await request.delete(`/api/note/hard-delete/${note_id}`);
      expect(res).to.have.status(200);
      expect(res.body).to.be.an("object");
      expect(res.body).to.property("is_deleted", true);
    });
  });
});

```

- The response status code from the endpoint should be `200`.
- For create, retrieve by ID and update, the response data should match the value that was passed in.
- For retrieve all, the response should be an array of data.
- For delete, the `is_deleted` value should be true.

Negative Test Cases

```
describe("Notes Negative Test Cases", () => {
  const invalid_note_id = "123"; // Invalid ID as it is not a valid ObjectID
  const valid_note_id = "613f4186abe8ac519b619877"; // Valid ObjectId but no such record in the DB

  describe("POST /", () => {
    it("Should not create a note", async () => {
      const res = await request.post("/api/note").send({ name: "Hello" });
      expect(res).to.have.status(422);
      expect(res.body).to.be.an("object");
      expect(res.body).haveOwnProperty("errors");
      expect(res.body).property("errors").to.deep.nested.property("title", ["The title field is required."]);
      expect(res.body)
        .property("errors")
        .to.deep.nested.property("description", ["The description field is required."]);
    });
  });

  describe("GET /", () => {
    it("Should not fetch a note due to invalid ID", async () => {
      const res = await request.get(`/api/note/${invalid_note_id}`);
      expect(res).to.have.status(422);
      expect(res.body).to.be.an("object");
      expect(res.body).haveOwnProperty("errors");
      expect(res.body).property("errors").to.deep.nested.property("note_id", ["The note id format is invalid."]);
    });

    it("Should not fetch a note due to record not found", async () => {
      const res = await request.get(`/api/note/${valid_note_id}`);
      expect(res).to.have.status(404);
      expect(res.body).to.be.an("object");
      expect(res.body).haveOwnProperty("errors");
      expect(res.body).property("errors", `Note ${valid_note_id} not found.`);
    });
  });

  describe("PUT /", () => {
    it("Should not update note due to invalid ID", async () => {
      const res = await request.put("/api/note").send({
        _id: invalid_note_id,
        title: "Update Note",
        description: "This note is updated in test",
      });
      expect(res).to.have.status(422);
      expect(res.body).to.be.an("object");
      expect(res.body).haveOwnProperty("errors");
      expect(res.body).property("errors").to.deep.nested.property("_id", ["The  id format is invalid."]);
    });

    it("Should not update note due to record not found", async () => {
      const res = await request.put("/api/note").send({
        _id: valid_note_id,
        title: "Update Note",
        description: "This note is updated in test",
      });
      expect(res).to.have.status(404);
      expect(res.body).to.be.an("object");
      expect(res.body).haveOwnProperty("errors");
      expect(res.body).property("errors", `Note ${valid_note_id} not found.`);
    });
  });

  describe("DELETE /", () => {
    it("Should not soft delete note due to invalid ID", async () => {
      const res = await request.delete(`/api/note/${invalid_note_id}`);
      expect(res).to.have.status(422);
      expect(res.body).to.be.an("object");
      expect(res.body).haveOwnProperty("errors");
      expect(res.body).property("errors").to.deep.nested.property("note_id", ["The note id format is invalid."]);
    });

    it("Should not soft delete note due to record not found", async () => {
      const res = await request.delete(`/api/note/${valid_note_id}`);
      expect(res).to.have.status(404);
```

```
        expect(res.body).to.be.an("object");
        expect(res.body).haveOwnProperty("errors");
        expect(res.body).property("errors", `Note ${valid_note_id} not found.`);
      });

      it("Should not hard delete note due to invalid ID", async () => {
        const res = await request.delete(`/api/note/hard-delete/${invalid_note_id}`);
        expect(res).to.have.status(422);
        expect(res.body).to.be.an("object");
        expect(res.body).haveOwnProperty("errors");
        expect(res.body).property("errors").to.deep.nested.property("note_id", ["The note id format is invalid."]);
      });

      it("Should not hard delete note due to record not found", async () => {
        const res = await request.delete(`/api/note/hard-delete/${valid_note_id}`);
        expect(res).to.have.status(404);
        expect(res.body).to.be.an("object");
        expect(res.body).haveOwnProperty("errors");
        expect(res.body).property("errors", `Note ${valid_note_id} not found.`);
      });
    });
  });

```

- The response status code should be either be `404` or `422`.
- For an attempt to create a note with invalid data, the status code should be `404` and an error message stating which fields are missing.
- For an attempt to retrieve, update, soft-delete, and hard-delete note with an invalid `_id` (ObjectId), the status code should be `422` and an error message stating the id format is invalid.
- For an attempt to retrieve, update, soft-delete and hard-delete note with a valid `_id` (ObjectId) but the record does not exist in the database, the status code should be `404` and an error message stating "Note {note_id} is not found."

Running the test through CI

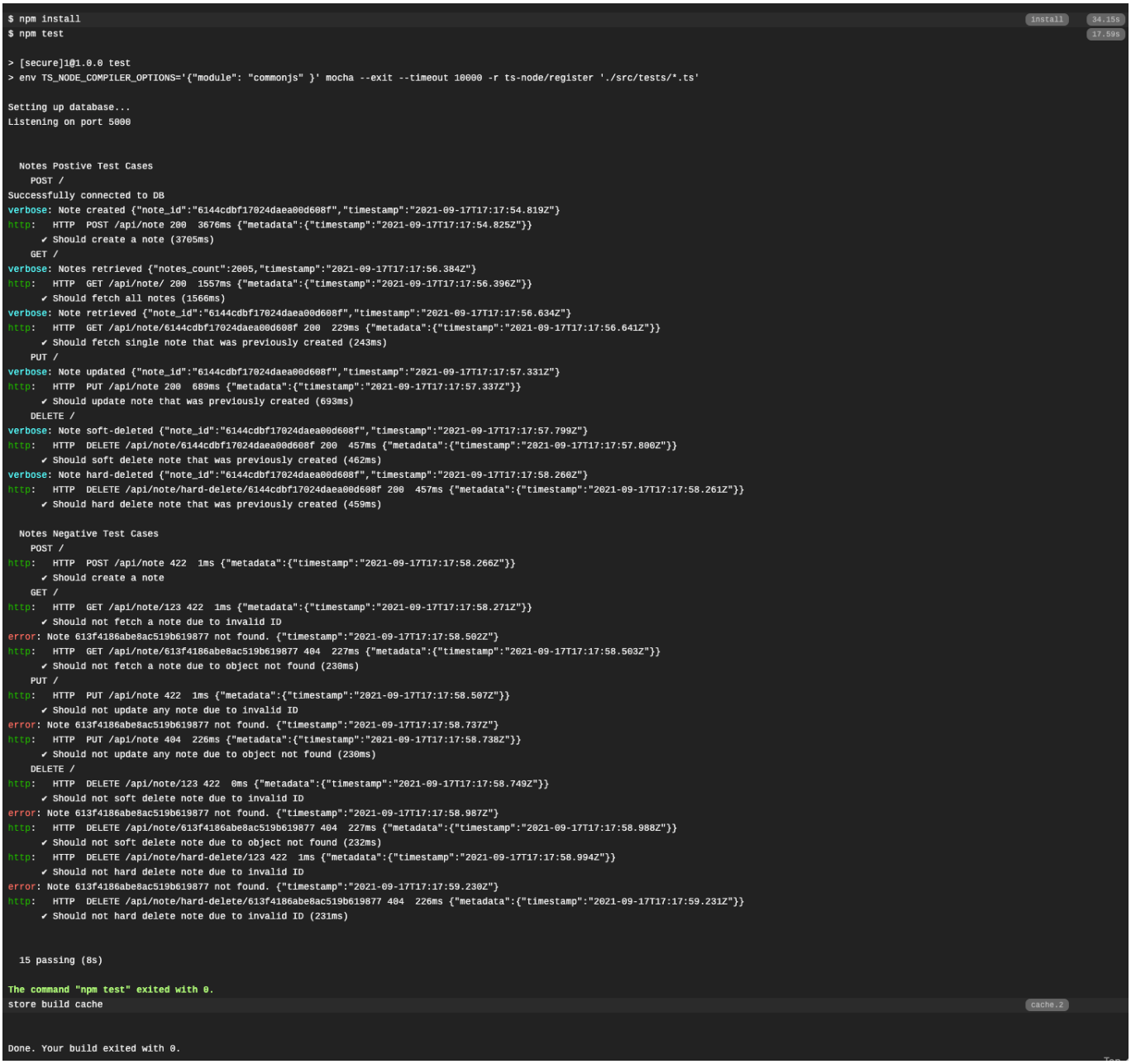
Code Snippet from .travis.yml

```yaml
language: node_js
node_js:
- node
cache:
  directories:
    - node_modules
before_install:
- cd backend
install:
- npm install
jobs:
  include:
  - stage: test
    script: npm run test
```

For this task, Travis has been integrated into the repository. Before it runs the jobs, it will perform `cd backend` and run `npm install` to install the necessary modules and libraries.

For CI, under jobs, a stage named `test` is created. `npm run test` is executed whenever the codes are pushed into the repository, which will execute the tests as defined above.

This is a screenshot of an example of the test.



References

- https://medium.com/@asciidev/testing-a-node-express-application-with-mocha-chai-9592d41c0083
- https://gist.github.com/cklanac/81a6f49fabb52b3c95dff397fe62c771

Task B3: Deployment through Continuous Deployment (CD)

Serverless Service: Serverless Google Cloud Functions

For this task, the backend server has been deployed using the Serverless Google Cloud Functions Provider.

Set Up

Travis

Code snippet from .travis.yml

```
- stage: deploy
  before_script: # Decrypt the API key required to deploy to Google Cloud Functions
  - openssl aes-256-cbc -K $encrypted_4e8c5512ae30_key -iv $encrypted_4e8c5512ae30_iv
    -in serverless-key.json.enc -out serverless-key.json -d
  script:
  - npm install -g serverless
  - npm run deploy
```

Continuing from the `.travis.yml` file that we have created in Task B2, we have defined a new job stage named `deploy`.

Before it runs `script`, it will decrypt the Google Cloud Application credentials key file credentials required for deployment. Then, it will install the `serverless` npm package then run the `npm run deploy` command, which is actually doing a `serverless deploy`.

Serverless

A `serverless.yml` has been set up as a set of instructions to deploy to Google Cloud Functions.

```yaml
service: cs3219-otot-task-b

provider:
  name: google
  stage: dev
  runtime: nodejs12
  project: cs3219-otot-task-b-325509
  credentials: ${env:TRAVIS_BUILD_DIR}/backend/serverless-key.json # <- the path must be absolute
  region: asia-southeast1
  environment: # Transfer the env variables from Travis to Google Cloud Functions
    MONGO_USERNAME: ${env:MONGO_USERNAME}
    MONGO_PASSWORD: ${env:MONGO_PASSWORD}
    MONGO_DB: ${env:MONGO_DB}

plugins:
  - serverless-plugin-typescript
  - serverless-google-cloudfunctions

package:
  exclude:
    - node_modules/**
    - .gitignore
    - .git/**

functions:
  app:
    handler: app
    runtime: nodejs12
    events:
      - http: path
```

As we are using Google's Service Account for deployment, it will use the credentials keyfile decrypted by Travis when the job `deploy` is run.

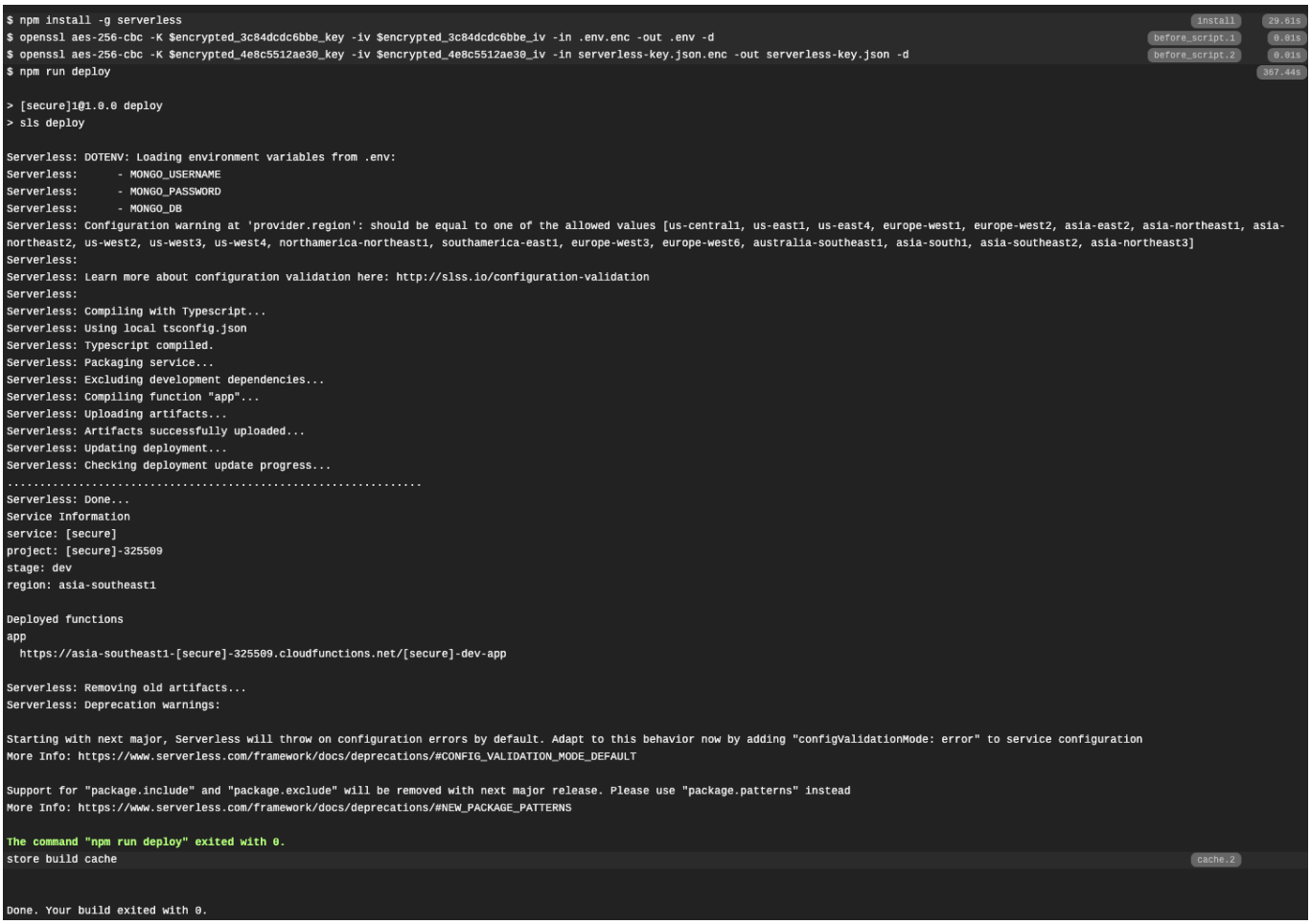
Afterwards, we will transfer the environment variables from Travis to Serverless Google Cloud Functions by exposing them under the provider's environment.

For function, we have exported our backend server as a variable named `app`, which will be invoked whenever the Google Cloud Functions endpoint is called.

- To test everything is working properly, we can also do a local deployment `npm run deploy` to ensure the configuration is properly set up.

Deploying through CD

This is a screenshot of an example of continuous deployment.



The application is deployed to https://asia-southeast1-cs3219-otot-task-b-325509.cloudfunctions.net/cs3219-otot-task-b-dev-app.

References

- https://www.serverless.com/framework/docs/providers/google/guide
- https://blog.travis-ci.com/2019-05-30-setting-up-a-ci-cd-process-on-github

Task B4: Implement a frontend

- **Frontend Framework**: Next.js (React.js)
- **UI Framework**: Material-UI
- **Additional Libraries Used**: Notistack and React Masonry CSS

This is an attempt in creating a frontend using Next.js using Material-UI to build a responsive web application.

The web application supports the CRUD operations created in Task B1.

To fulfil the learning objectives, the codes are structured in an MVC folder structure (which might not be conventional), along with using React's useReducer as a store.

Please ensure you are in the `frontend` directory (`cd frontend`).

Please create a `.env` file with the following variables:

```
DB_HOST_URL = "https://asia-southeast1-cs3219-otot-task-b-
325509.cloudfunctions.net/cs3219-otot-task-b-dev-app"
```

Install the necessary modules

```
npm install
```

Start the server

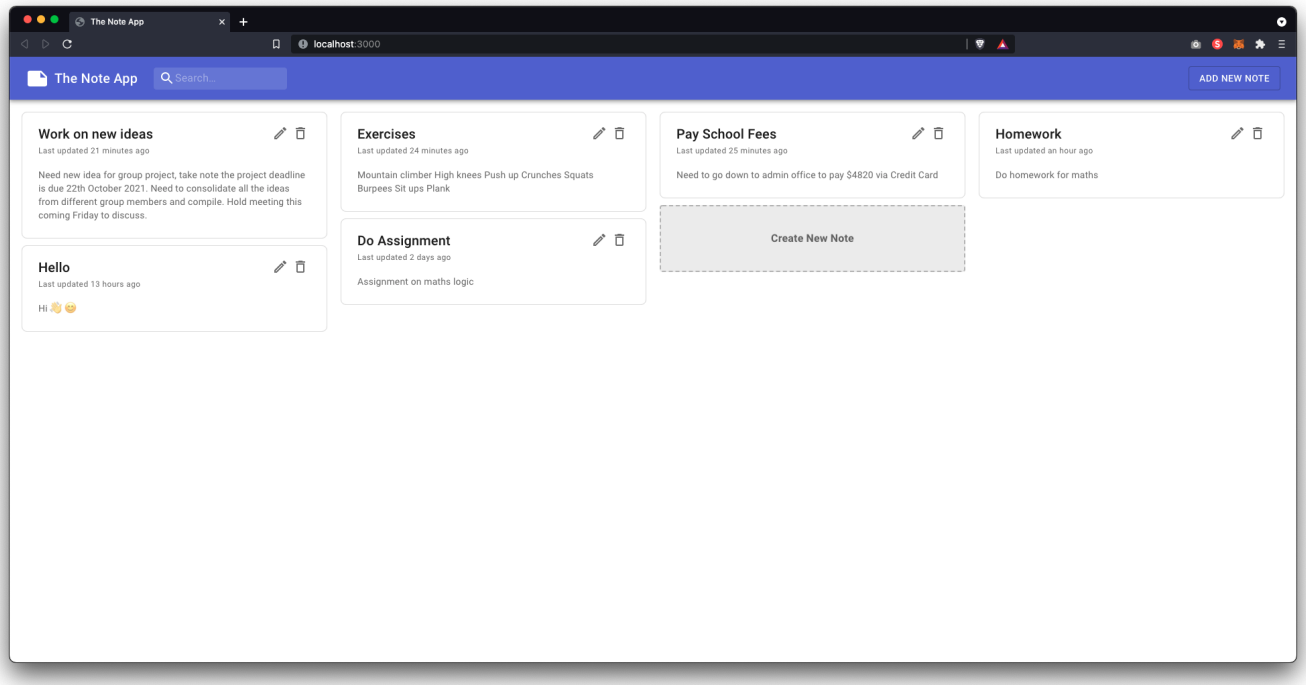
```
npm run dev
```

Endpoint

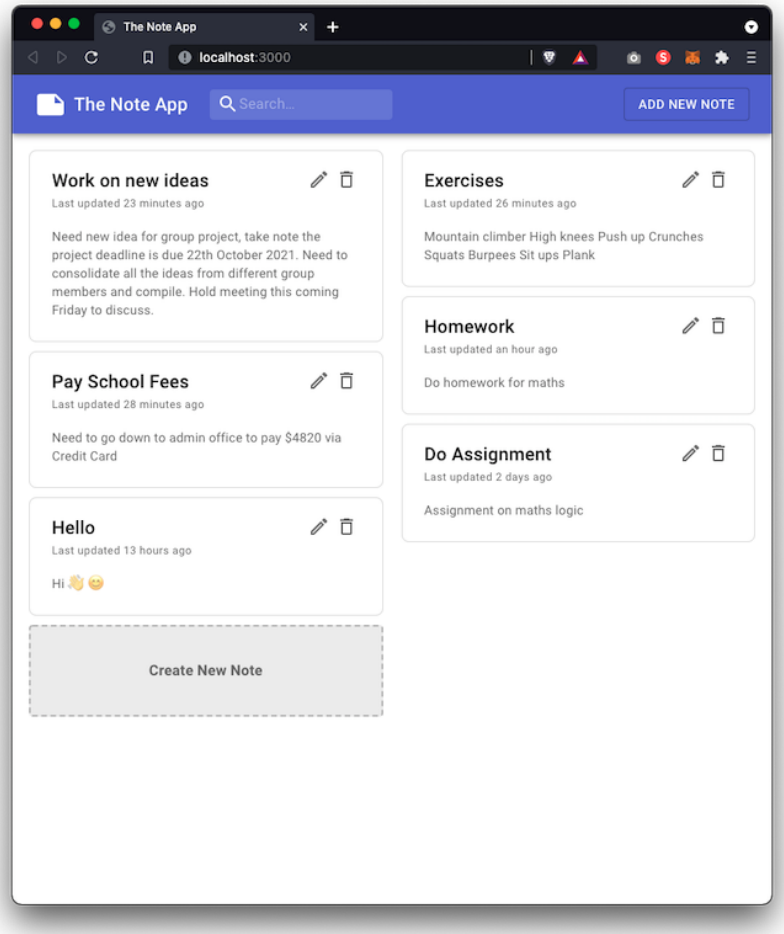
- Localhost: http://localhost:3000
- Deployed Endpoint: https://cs3219-otot-task-b-325509.as.r.appspot.com/

Demonstration

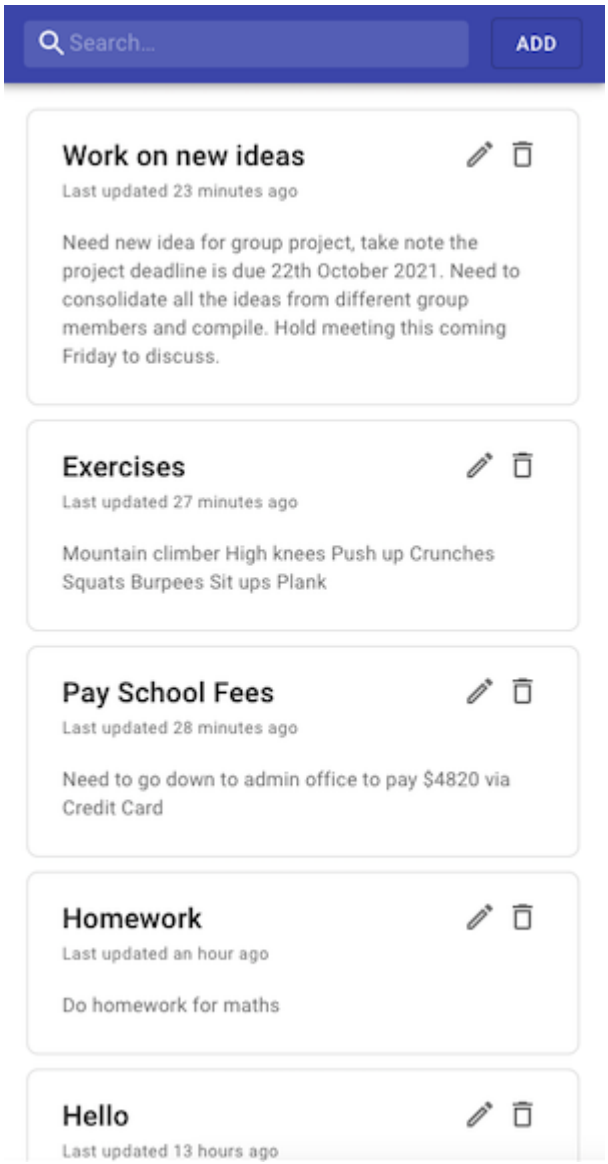
Desktop View



Tablet / Smaller Screen View



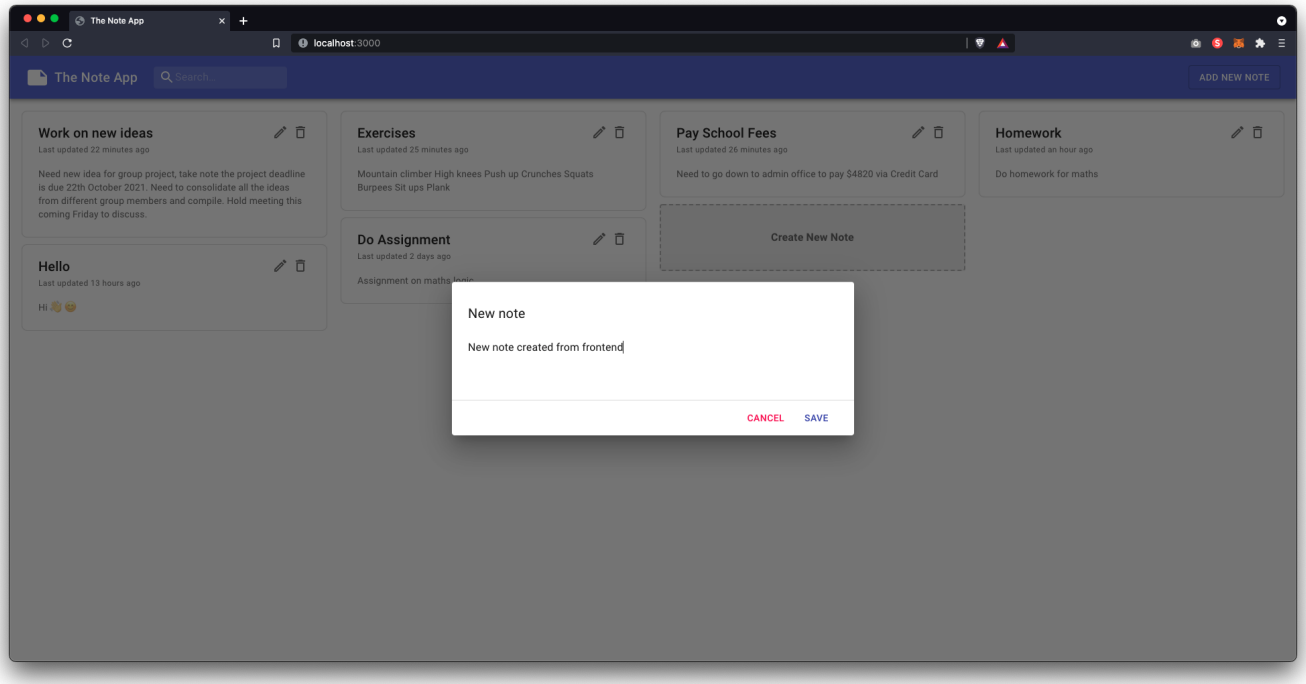
Mobile View


Search...
ADD
Work on new ideas
Last updated 23 minutes ago
Need new idea for group project, take note the project deadline is due 22th October 2021. Need to consolidate all the ideas from different group members and compile. Hold meeting this coming Friday to discuss.
Exercises
Last updated 27 minutes ago
Mountain climber High knees Push up Crunches Squats Burpees Sit ups Plank
Pay School Fees
Last updated 28 minutes ago
Need to go down to admin office to pay $4820 via Credit Card
Homework
Last updated an hour ago
Do homework for maths
Hello
Last updated 13 hours ago

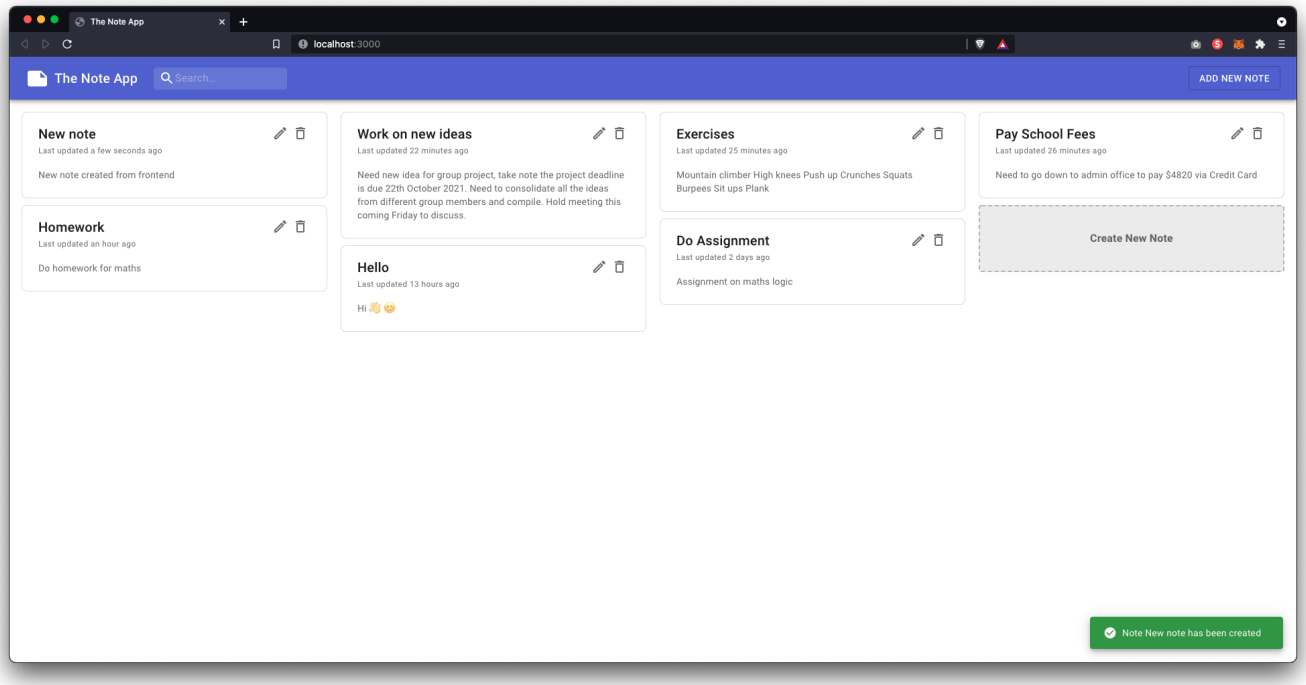
Create

- You can open the create dialog by either pressing `Add New Note` on the top right or the `Create New Note` placeholder card in the masonry grid.

Desktop View

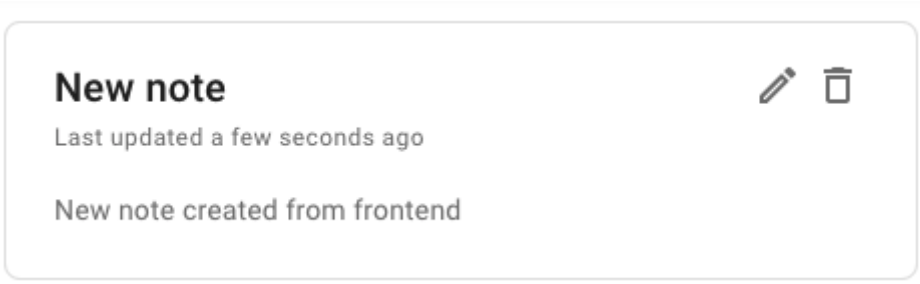


- After the note is created, a notification will appear on the bottom right of the screen, and the new note is displayed as the first item in the masonry grid.

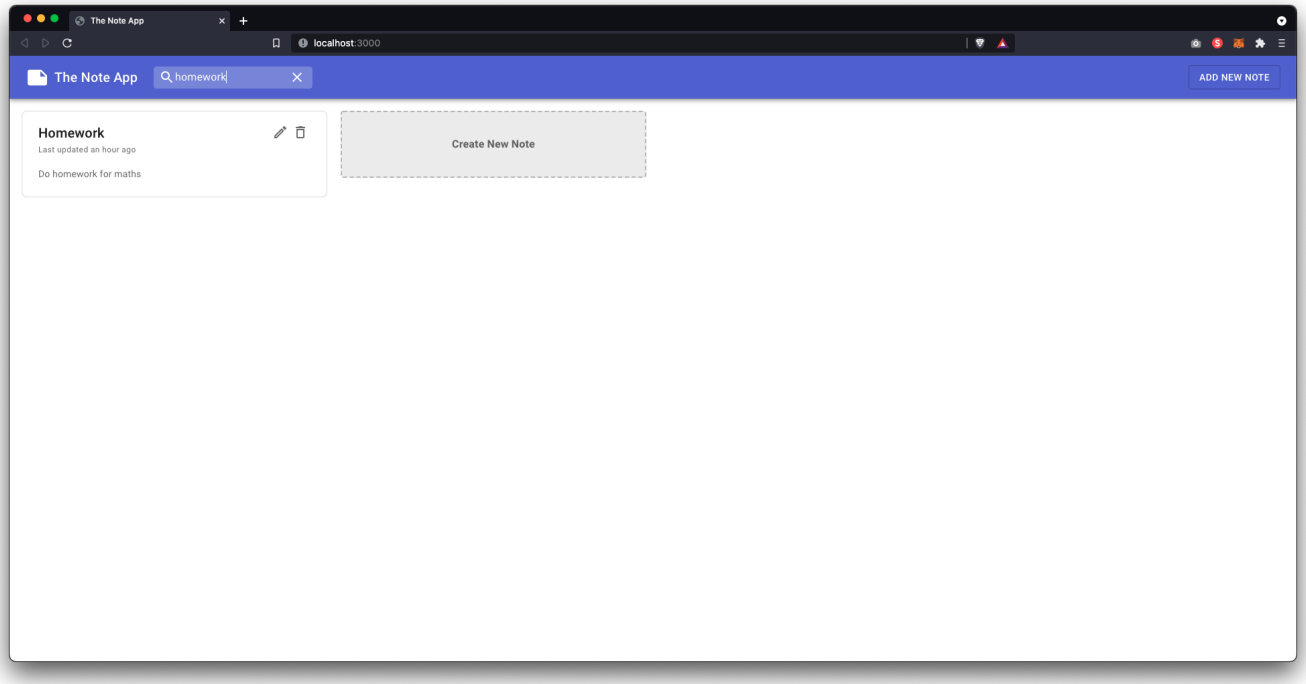


Retrieve

- Each note card consist of `title`, `description` and last `updated_at` information.
- There is an edit and delete button to the right of the card.

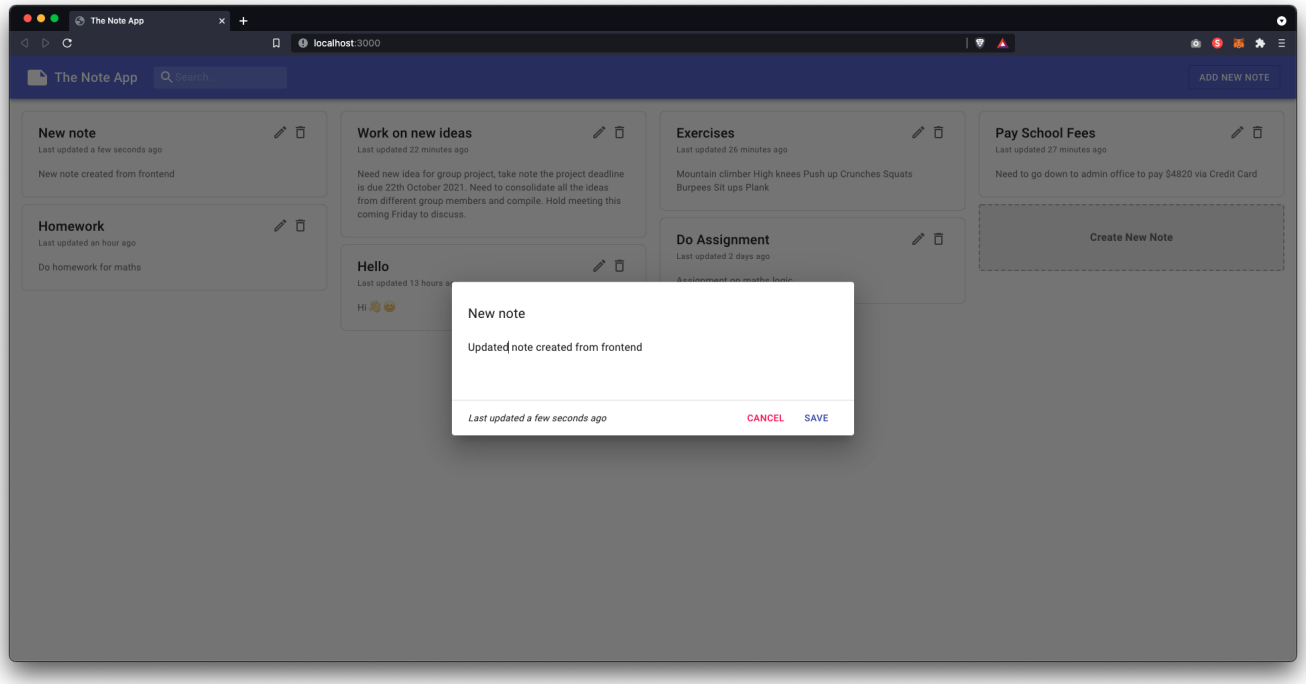


- You can perform searching (server-side filtering) by typing to the search box in the app bar.

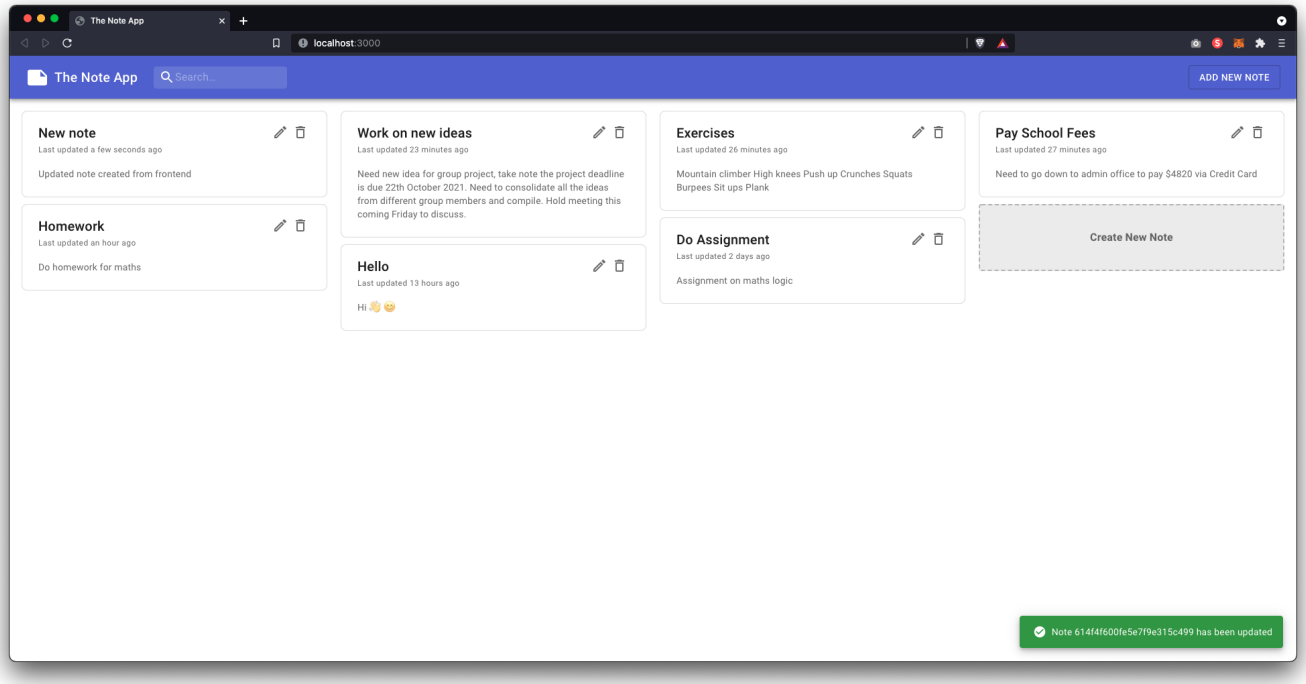


Update

- Click the edit (pencil) button on the card, and the update dialog should appear.

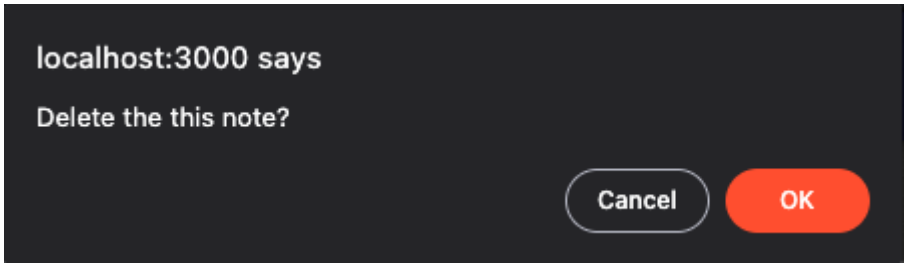


- After the note is created, a notification will appear on the bottom right of the screen, and the updated note is displayed as the first item in the masonry grid.

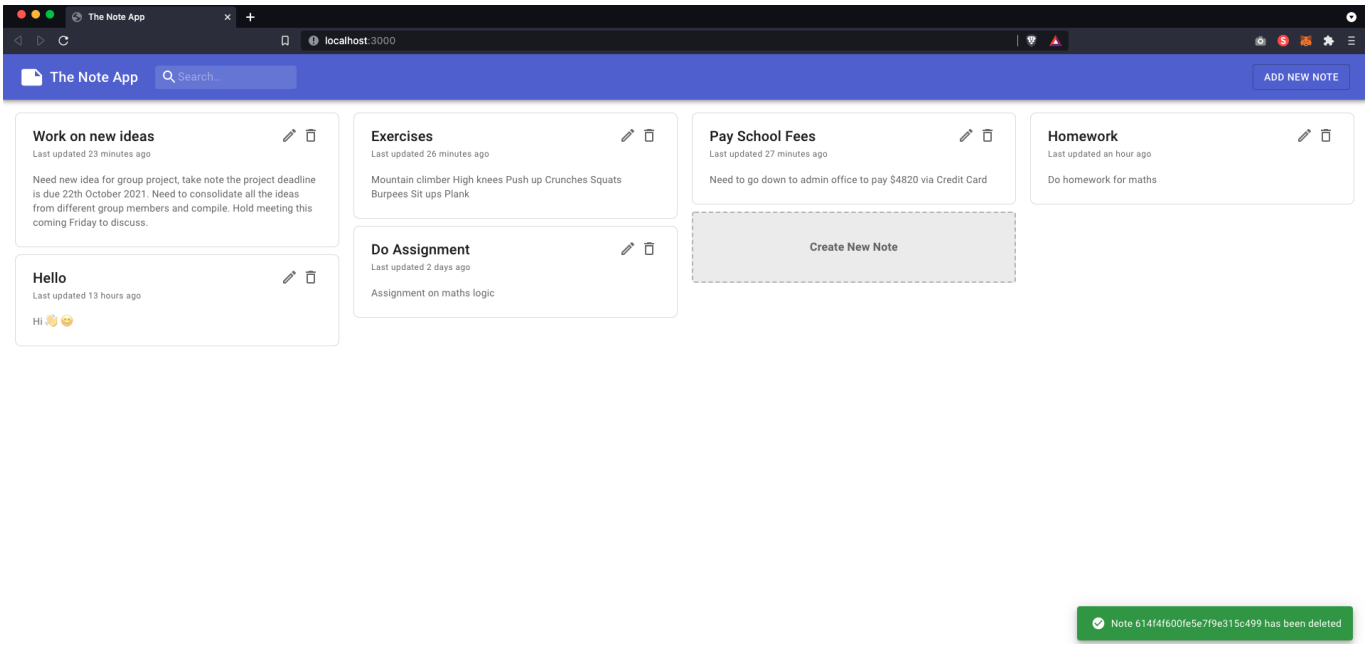


Delete

- Click the delete (trash) button on the card, and a delete warning should appear.



- After the note is deleted, a notification will appear on the bottom right of the screen, and the note is removed from the masonry grid.



Learning Outcome

- Despite the learning objectives being to use an MVC in real-life frameworks, the newer frameworks are no longer based on the "MVC" structure.
- The closest thing to MVC was Redux, which goes from Reducer -> Store -> View.
- In this task, I tried to replicate as closely as possible by utilizing controllers (which will interact with the APIs) and models (to define the attributes) while using React's reducer and store functionality.

References

- https://nextjs.org/learn/basics/create-nextjs-app
- https://hmh.engineering/using-react-contextapi-usereducer-as-a-replacement-of-redux-as-a-state-management-architecture-336452b2930e