

CS3219 Task B: CRUD Application Task

Name: Koh Vinleon

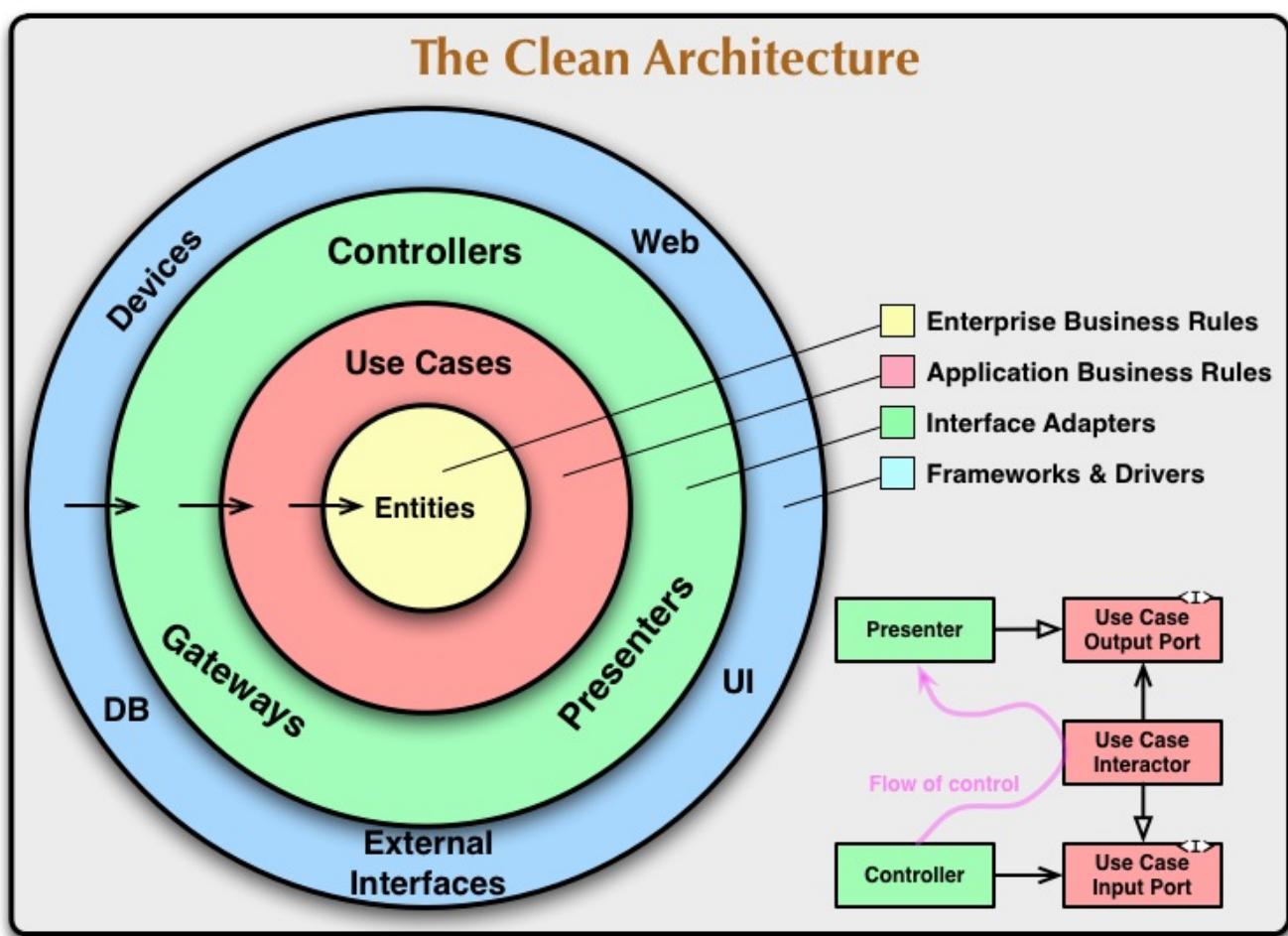
Matric Number: A0202155W

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:/^[\d-a-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)

The screenshot shows the Postman interface with a successful API call. The request URL is `http://localhost:5000/api/note/`. The response body is a JSON object:

```
1 {  
2   "_id": "6147ecd8bdf3052cc4aab5cc",  
3   "title": "Hello",  
4   "description": "Hello This Is A New Note",  
5   "created_at": "2021-09-20T02:07:20.972Z",  
6   "updated_at": "2021-09-20T02:07:20.972Z"  
7 }  
8  
9 }
```

- 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.

The screenshot shows the Postman interface with a test script in the 'Tests' tab:

```
1 var jsonData = JSON.parse(responseBody);  
2 postman.setEnvironmentVariable("note_id", jsonData.data._id);  
3
```

The right sidebar shows the 'Test scripts' section with a note about running scripts after the response is received.

Error (422)

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

The screenshot shows a Postman request to `http://localhost:5000/api/note/`. The body contains a single key-value pair: `"title": "Hello"`. The response status is `422 Unprocessable Entity`, with the error message: `"The description field is required."`.

```
1
2 ...
3 ... "title": "Hello"
4
```

```
1
2 ...
3 ...
4
5 ...
6 ...
7
```

Status: 422 Unprocessable Entity Time: 19 ms Size: 350 B Save Response

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

The screenshot shows a Postman request to `http://localhost:5000/api/note/`. The body contains two key-value pairs: `"title": 123` and `"description": "Description"`. The response status is `422 Unprocessable Entity`, with the error message: `"The title must be a string."`.

```
1 ...
2 ...
3 ...
4
```

```
1
2 ...
3 ...
4
```

Status: 422 Unprocessable Entity Time: 16 ms Size: 337 B Save Response

- 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)

The screenshot shows a Postman request for `http://localhost:5000/api/note`. The 'Params' tab is selected, showing a single query parameter `Key` with value `Value`. The 'Body' tab displays the JSON response:

```
1 "data": [
2   {
3     "_id": "6147ecd8bf3052cd4aab5cc",
4     "title": "Hello",
5     "description": "Hello This Is A New Note",
6     "created_at": "2021-09-20T02:07:20.972Z",
7     "updated_at": "2021-09-20T02:07:20.972Z"
8   },
9   {
10     "_id": "6146de62ac43d74496fa7daf",
11     "title": "Hello",
12     "description": "Hello This Is A New Note",
13     "created_at": "2021-09-19T06:53:22.172Z",
14     "updated_at": "2021-09-19T06:53:22.172Z"
15   }
16 ]
17 ]
```

The status bar at the bottom indicates `Status: 200 OK Time: 40 ms Size: 623 B`.

- 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`.

The screenshot shows a Postman request for `http://localhost:5000/api/note?query=homework`. The 'Params' tab is selected, showing a query parameter `query` with value `homework`. The 'Body' tab displays the JSON response:

```
1 "data": [
2   {
3     "_id": "614cc3e0e4c87d14ce00dd57",
4     "title": "Homework",
5     "description": "Do homework for maths",
6     "created_at": "2021-09-23T18:13:52.388Z",
7     "updated_at": "2021-09-25T15:41:18.744Z"
8   }
9 ]
10 ]
```

The screenshot shows a Postman request for `http://localhost:5000/api/note?query=homework`. The 'Params' tab is selected, showing a query parameter `query` with value `homework`. The 'Body' tab displays the JSON response:

```
1 "data": []
2 ]
```

- 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)

The screenshot shows a Postman collection named "CS3219-TaskB". A new request is being created for "Get Note" with the method set to "GET" and the URL as "http://localhost:5000/api/note/({{note_id}})". The "Params" tab is selected, showing a single query parameter "Key" with "Value". The "Body" tab shows the response body in JSON format:

```
1 "data": {  
2   "_id": "6147ecd8bdf3052cc4aab5cc",  
3   "title": "Hello",  
4   "description": "Hello This Is A New Note",  
5   "created_at": "2021-09-20T02:07:20.972Z",  
6   "updated_at": "2021-09-20T02:07:20.972Z"  
7 }  
8  
9
```

The status bar at the bottom indicates "Status: 200 OK Time: 39 ms Size: 448 B".

Error (404)

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

The screenshot shows a Postman request for "Get Note" with the URL as "http://localhost:5000/api/note/6147f0dff6e558b2a6b24fa8". The "Params" tab is selected, showing a single query parameter "Key" with "Value". The "Body" tab shows the response body in JSON format:

```
1 "errors": "Note 6147f0dff6e558b2a6b24fa8 not found."  
2  
3
```

The status bar at the bottom indicates "Status: 404 Not Found Time: 69 ms Size: 327 B".

Error (422)

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

The screenshot shows a Postman interface with the following details:

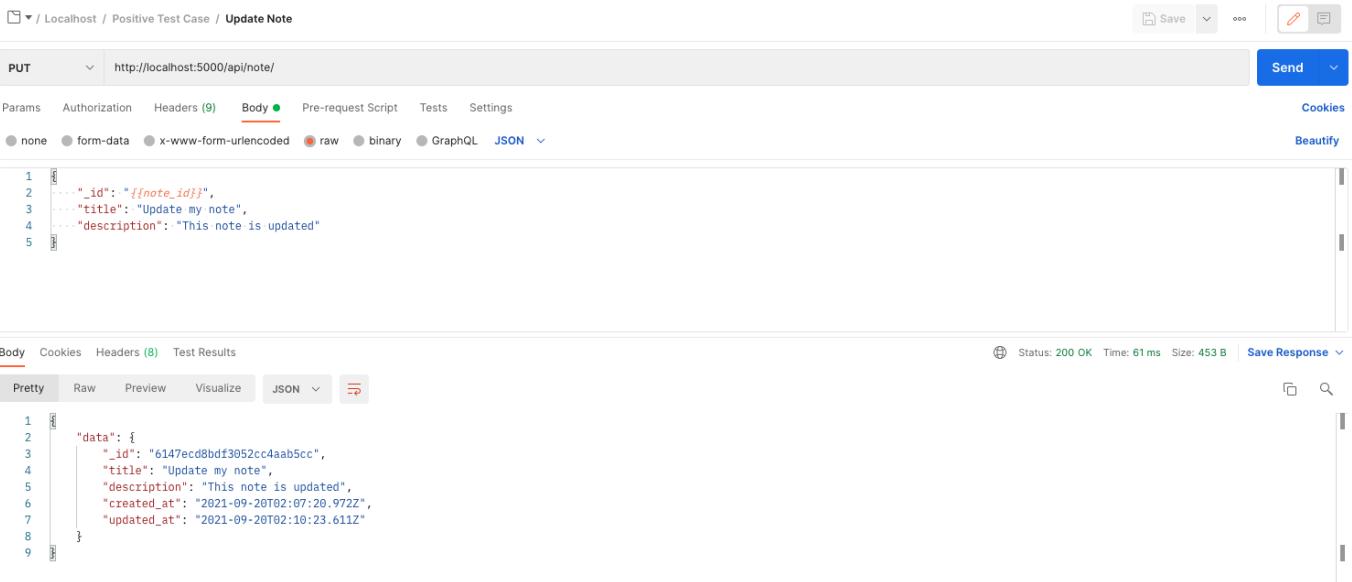
- Request URL:** GET http://localhost:5000/api/note/123
- Headers:** Authorization, Headers (7), Body, Pre-request Script, Tests, Settings
- Query Params:** A table with one row: Key (Key) and Value (Value). Description (Description) is also present.
- Body:** Cookies, Headers (8), Test Results. The Body tab is selected.
- Response:** Status: 422 Unprocessable Entity, Time: 12 ms, Size: 342 B. The response body is displayed in JSON format:

```
1 | {
2 |   "errors": {
3 |     "note_id": [
4 |       "The note id format is invalid."
5 |     ]
6 |   }
7 | }
```

PUT (Update)

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

Success (200)



The screenshot shows a Postman request to `http://localhost:5000/api/note/`. The request method is **PUT**. The JSON body contains:

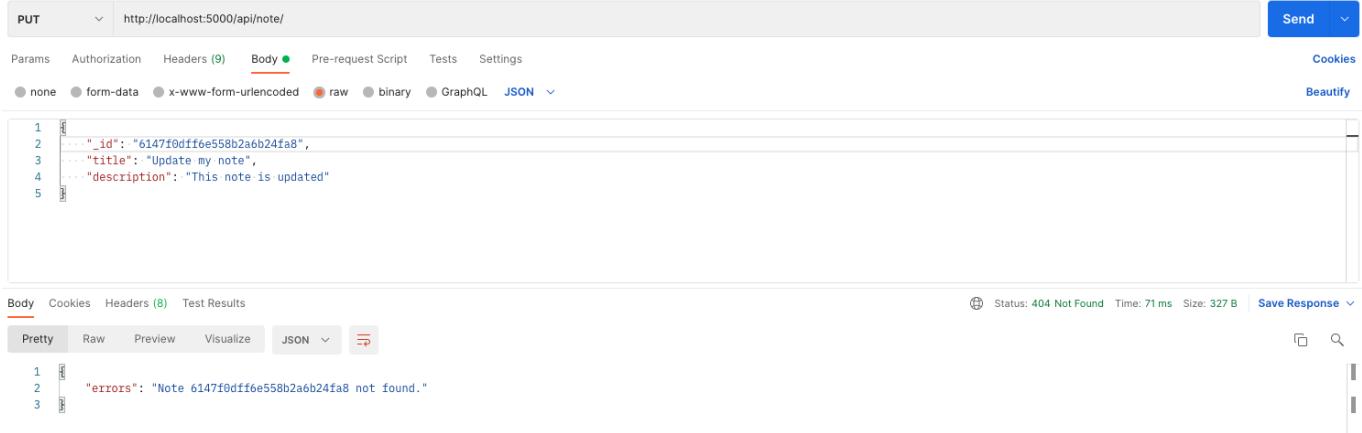
```
1 ... "_id": "{{note_id}}",
2 ... "title": "Update my note",
3 ... "description": "This note is updated"
```

The response status is **200 OK**, with a response body:

```
1 ...
2 "data": {
3   "_id": "6147ecd8bdf3052cc4aab5cc",
4   "title": "Update my note",
5   "description": "This note is updated",
6   "created_at": "2021-09-20T02:07:20.972Z",
7   "updated_at": "2021-09-20T02:10:23.611Z"
8 }
```

Error (404)

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



The screenshot shows a Postman request to `http://localhost:5000/api/note/`. The request method is **PUT**. The JSON body contains:

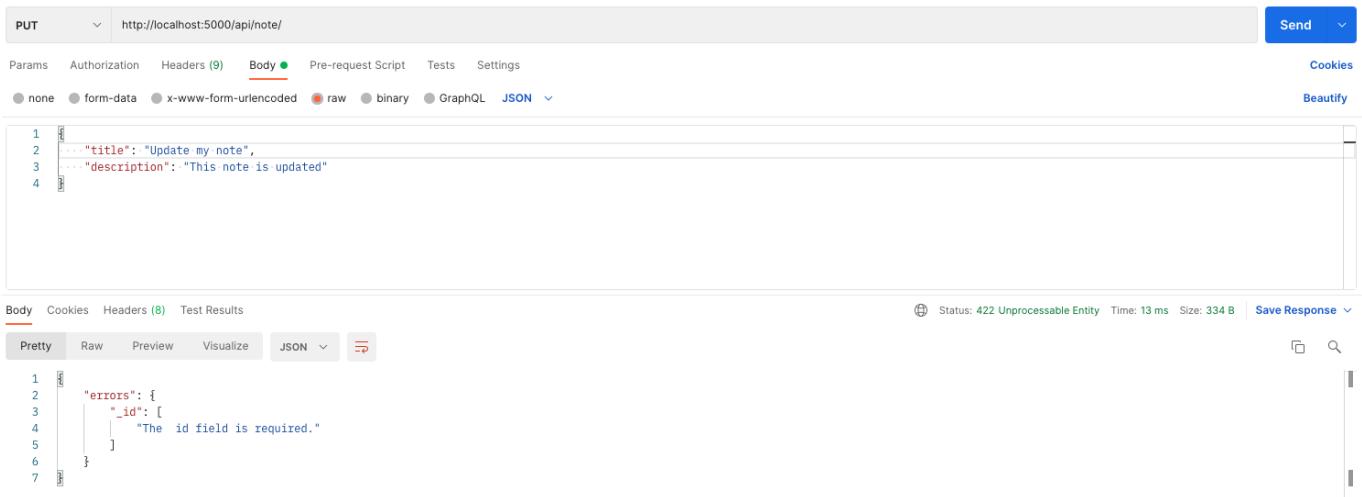
```
1 ...
2 ... "_id": "6147f0dff6e558b2a6b24fa8",
3 ... "title": "Update my note",
4 ... "description": "This note is updated"
```

The response status is **404 Not Found**, with a response body:

```
1 ...
2 "errors": "Note 6147f0dff6e558b2a6b24fa8 not found."
```

Error (422)

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



PUT http://localhost:5000/api/note/

Params Authorization Headers (9) Body **JSON** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2   "title": "Update my note",
3   "description": "This note is updated"
4
```

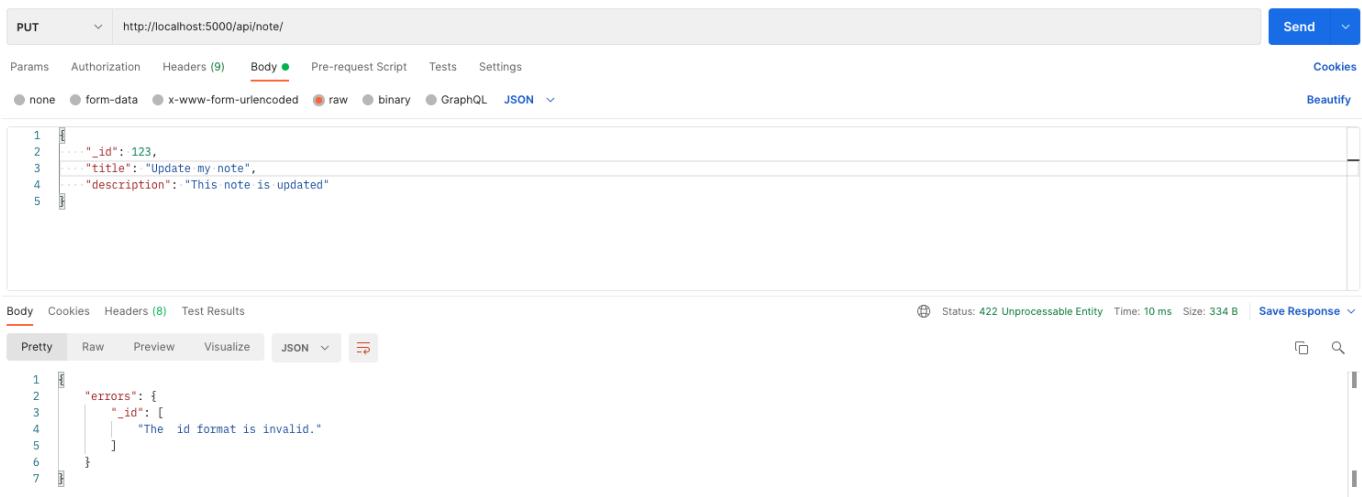
Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1
2   "errors": [
3     "_id": [
4       "The id field is required."
5     ]
6   ]
7
```

Status: 422 Unprocessable Entity Time: 13 ms Size: 334 B Save Response

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



PUT http://localhost:5000/api/note/

Params Authorization Headers (9) Body **JSON** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

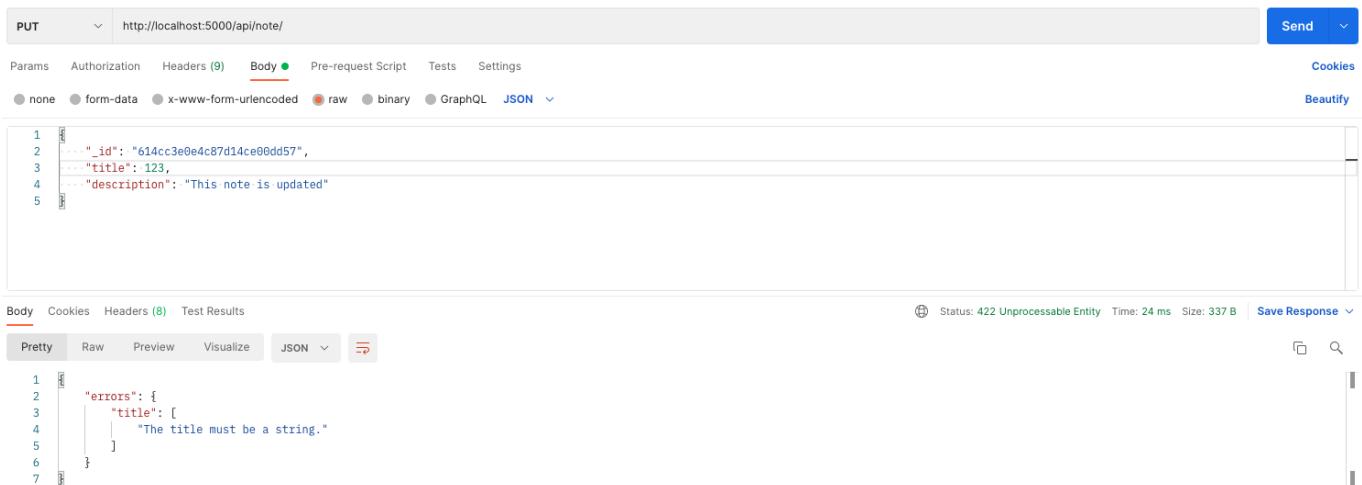
```
1
2   "_id": 123,
3   "title": "Update my note",
4   "description": "This note is updated"
5
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1
2   "errors": [
3     "_id": [
4       "The id format is invalid."
5     ]
6   ]
7
```

Status: 422 Unprocessable Entity Time: 10 ms Size: 334 B Save Response



PUT http://localhost:5000/api/note/

Params Authorization Headers (9) Body **JSON** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2   "_id": "614cc3e0e4c87d14ce00dd57",
3   "title": 123,
4   "description": "This note is updated"
5
```

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

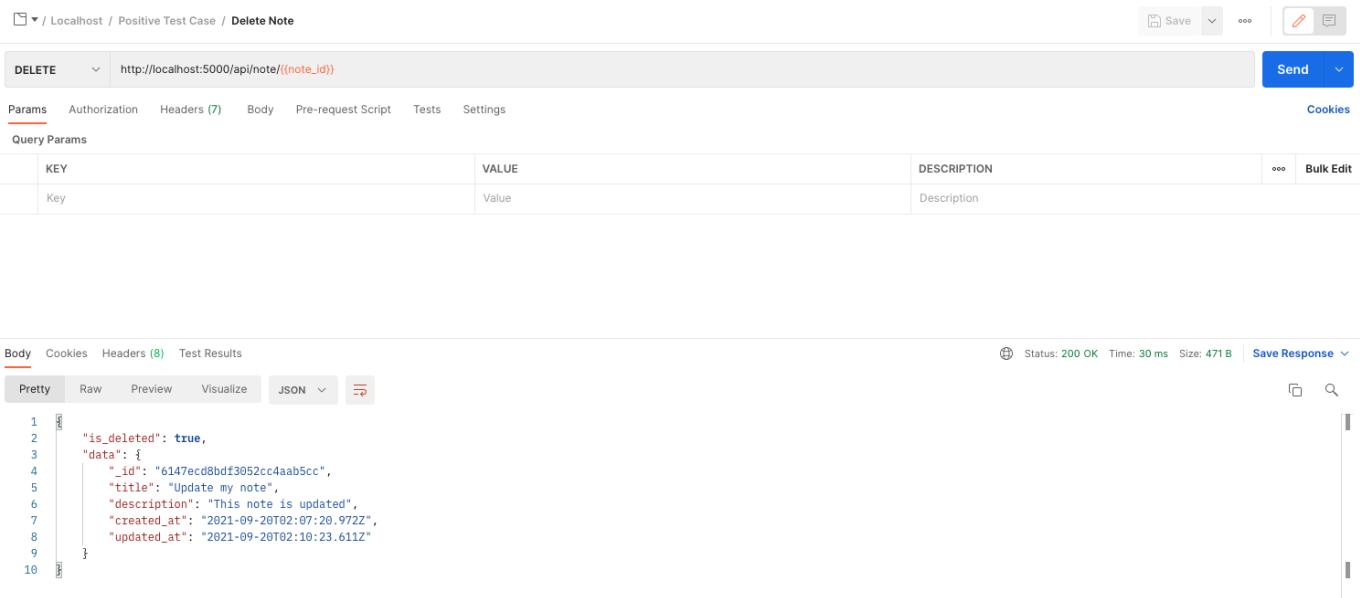
```
1
2   "errors": [
3     "title": [
4       "The title must be a string."
5     ]
6   ]
7
```

Status: 422 Unprocessable Entity Time: 24 ms Size: 337 B Save Response

Delete (Soft Delete)

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

Success (200)

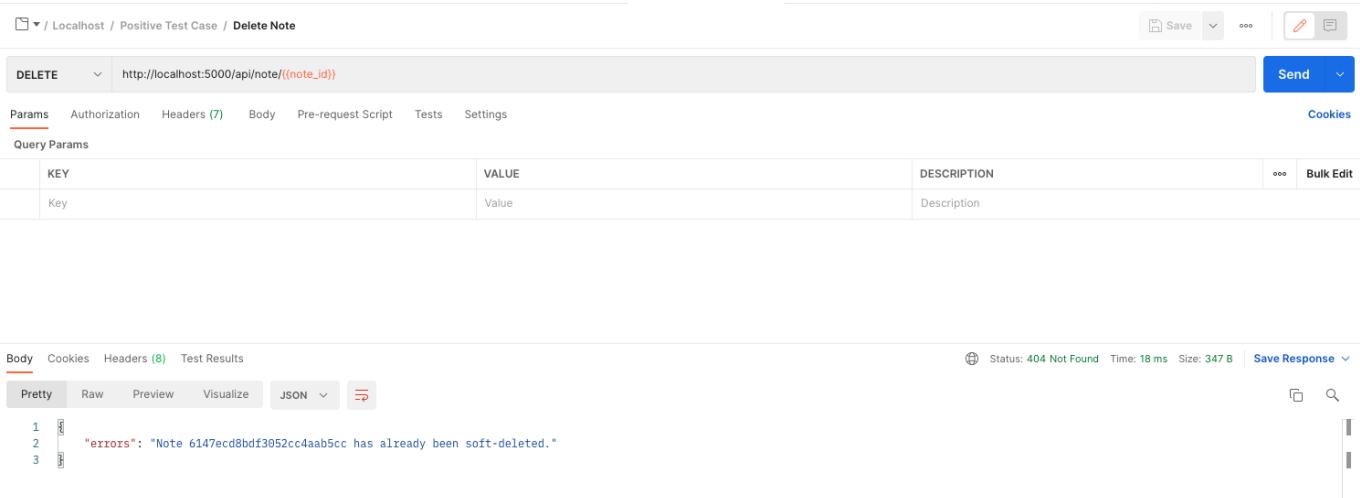


The screenshot shows a Postman request to `http://localhost:5000/api/note/6147ecd8bd3052cc4aab5cc`. The response status is **200 OK** with a time of **30 ms** and a size of **471 B**. The response body is:

```
1 "is_deleted": true,
2 "data": {
3     "_id": "6147ecd8bd3052cc4aab5cc",
4     "title": "Update my note",
5     "description": "This note is updated",
6     "created_at": "2021-09-20T02:07:20.972Z",
7     "updated_at": "2021-09-20T02:10:23.611Z"
8 }
```

Error (404)

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



The screenshot shows a Postman request to `http://localhost:5000/api/note/6147ecd8bd3052cc4aab5cc`. The response status is **404 Not Found** with a time of **18 ms** and a size of **347 B**. The response body is:

```
1 "errors": "Note 6147ecd8bd3052cc4aab5cc has already been soft-deleted."
```

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

DELETE http://localhost:5000/api/note/6147f0dff6e558b2a6b24fa8

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1 "errors": "Note 6147f0dff6e558b2a6b24fa8 not found."
2
3

```

Status: 404 Not Found Time: 39 ms Size: 327 B Save Response

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 http://localhost:5000/api/note/123

Params Authorization Headers (7) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1 "errors": {
2   "note_id": [
3     "The note id format is invalid."
4   ]
5 }
6
7

```

Status: 422 Unprocessable Entity Time: 26 ms Size: 342 B Save Response

Delete (Hard Delete)

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

Success (200)

The screenshot shows a Postman test case titled "Hard Delete Note". The method is set to "DELETE" and the URL is "http://localhost:5000/api/note/hard-delete/((note_id))". A query parameter "Key" is added with the value "Value". The response status is 200 OK, time is 38 ms, size is 286 B, and the response body is {"is_deleted": true}.

Error (404)

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

The screenshot shows a Postman test case titled "Hard Delete Note". The method is set to "DELETE" and the URL is "http://localhost:5000/api/note/hard-delete/((note_id))". A query parameter "Key" is added with the value "Value". The response status is 404 Not Found, time is 23 ms, size is 327 B, and the response body is {"errors": "Note 6147ecd8bd3052cc4aab5cc not found."}

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.

The screenshot shows a Postman interface with a DELETE request to `http://localhost:5000/api/note/hard-delete/123`. The 'Params' tab is selected. In the 'Query Params' table, there is one entry: 'Key' (Value) and 'Description' (Description). The 'Body' tab is selected, showing a JSON response with a status of 422 Unprocessable Entity. The response body is:

```
1 "errors": {  
2   "note_id": [  
3     "The note id format is invalid."  
4   ]  
5 }  
6  
7 }
```

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.

Positive Test Cases

```
1  describe("Notes Positive Test Cases", () => {
2    let note_id = ""; // Used in retrieval & deletion test
3    const title = "New note from test";
4    const description = "Note created using test";
5    const updated_title = "Updated note from test";
6    const updated_description = "This note is updated in test";
7
8    describe("POST /", () => {
9      it("Should create a note", async () => {
10        const res = await request.post("/api/note").send({ title, description });
11        expect(res).to.have.status(200);
12        expect(res.body).to.be.an("object");
13        expect(res.body).to.haveOwnProperty("data");
14        expect(res.body).to.property("data").property("title", title);
15        expect(res.body).to.property("data").property("description", description);
16        note_id = _.get(res.body, "data._id");
17      });
18    });
19
20    describe("GET /", () => {
21      it("Should fetch all notes", async () => {
22        const res = await request.get("/api/note/");
23        expect(res).to.have.status(200);
24        expect(res.body).to.be.an("object");
25        expect(res.body).to.haveOwnProperty("data");
26        expect(_.get(res.body, "data")).to.be.an("array");
27      });
28
29      it("Should fetch single note that was previously created", async () => {
30        const res = await request.get(`/api/note/${note_id}`);
31        expect(res).to.have.status(200);
32        expect(res.body).to.be.an("object");
33        expect(res.body).to.haveOwnProperty("data");
34        expect(res.body).to.property("data").property("title", title);
35        expect(res.body).to.property("data").property("description", description);
36      });
37    });
38
39    describe("PUT /", () => {
40      it("Should update note that was previously created", async () => {
41        const res = await request.put("/api/note").send({
42          _id: note_id,
43          title: updated_title,
```

```
44     description: updated_description,
45   });
46   expect(res).to.have.status(200);
47   expect(res.body).to.be.an("object");
48   expect(res.body).to.haveOwnProperty("data");
49   expect(res.body).to.property("data").property("title", updated_title);
50   expect(res.body).to.property("data").property("description", updated_description);
51 });
52 });
53
54 describe("DELETE /", () => {
55   it("Should soft delete note that was previously created", async () => {
56     const res = await request.delete(`api/note/${note_id}`);
57     expect(res).to.have.status(200);
58     expect(res.body).to.be.an("object");
59     expect(res.body).to.property("is_deleted", true);
60   });
61
62   it("Should hard delete note that was previously created", async () => {
63     const res = await request.delete(`api/note/hard-delete/${note_id}`);
64     expect(res).to.have.status(200);
65     expect(res.body).to.be.an("object");
66     expect(res.body).to.property("is_deleted", true);
67   });
68 });
69 });
70
```

- 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

```

1 describe("Notes Negative Test Cases", () => {
2   const invalid_note_id = "123"; // Invalid ID as it is not a valid ObjectId
3   const valid_note_id = "613f4186abe8ac519b619877"; // Valid ObjectId but no such record in the DB
4
5   describe("POST /", () => {
6     it("Should not create a note", async () => {
7       const res = await request.post("/api/note").send({ name: "Hello" });
8       expect(res).to.have.status(422);
9       expect(res.body).to.be.an("object");
10      expect(res.body).haveOwnProperty("errors");
11      expect(res.body.property("errors").to.deep.nested.property("title", ["The title field is required."]));
12      expect(res.body)
13        .property("errors")
14          .to.deep.nested.property("description", ["The description field is required."]);
15    });
16  });
17
18  describe("GET /", () => {
19    it("Should not fetch a note due to invalid ID", async () => {
20      const res = await request.get(`/api/note/${invalid_note_id}`);
21      expect(res).to.have.status(422);
22      expect(res.body).to.be.an("object");
23      expect(res.body).haveOwnProperty("errors");
24      expect(res.body.property("errors").to.deep.nested.property("note_id", ["The note id format is invalid."]));
25    });
26
27    it("Should not fetch a note due to record not found", async () => {
28      const res = await request.get(`api/note/${valid_note_id}`);
29      expect(res).to.have.status(404);
30      expect(res.body).to.be.an("object");
31      expect(res.body).haveOwnProperty("errors");
32      expect(res.body.property("errors", `Note ${valid_note_id} not found.`));
33    });
34  });
35
36  describe("PUT /", () => {
37    it("Should not update note due to invalid ID", async () => {
38      const res = await request.put("/api/note").send({
39        _id: invalid_note_id,
40        title: "Update Note",
41        description: "This note is updated in test",
42      });
43      expect(res).to.have.status(422);
44      expect(res.body).to.be.an("object");
45      expect(res.body).haveOwnProperty("errors");
46      expect(res.body.property("errors").to.deep.nested.property("_id", ["The id format is invalid."]));
47    });
48
49    it("Should not update note due to record not found", async () => {
50      const res = await request.put("/api/note").send({
51        _id: valid_note_id,
52        title: "Update Note",
53        description: "This note is updated in test",
54      });
55      expect(res).to.have.status(404);
56      expect(res.body).to.be.an("object");
57      expect(res.body).haveOwnProperty("errors");
58      expect(res.body.property("errors", `Note ${valid_note_id} not found.`));
59    });
60  });
61
62  describe("DELETE /", () => {
63    it("Should not soft delete note due to invalid ID", async () => {
64      const res = await request.delete(`api/note/${invalid_note_id}`);
65      expect(res).to.have.status(422);
66      expect(res.body).to.be.an("object");
67      expect(res.body).haveOwnProperty("errors");
68      expect(res.body.property("errors").to.deep.nested.property("note_id", ["The note id format is invalid."]));
69    });
70
71    it("Should not soft delete note due to record not found", async () => {
72      const res = await request.delete(`api/note/${valid_note_id}`);
73      expect(res).to.have.status(404);
74    });
75  });

```

```

73     expect(res).to.have.status(404);
74     expect(res.body).to.be.an("object");
75     expect(res.body).haveOwnProperty("errors");
76     expect(res.body).property("errors", `Note ${valid_note_id} not found.`);
77   });
78
79   it("Should not hard delete note due to invalid ID", async () => {
80     const res = await request.delete(`/api/note/hard-delete/${invalid_note_id}`);
81     expect(res).to.have.status(422);
82     expect(res.body).to.be.an("object");
83     expect(res.body).haveOwnProperty("errors");
84     expect(res.body).property("errors").to.deep.nested.property("note_id", ["The note id format is invalid."]);
85   });
86
87   it("Should not hard delete note due to record not found", async () => {
88     const res = await request.delete(`/api/note/hard-delete/${valid_note_id}`);
89     expect(res).to.have.status(404);
90     expect(res.body).to.be.an("object");
91     expect(res.body).haveOwnProperty("errors");
92     expect(res.body).property("errors", `Note ${valid_note_id} not found.`);
93   });
94 });
95 });
96

```

- 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](#)

```
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.

```
$ npm install
$ npm test

> [secure]1@1.0.0 test
> env TS_NODE_COMPILER_OPTIONS='{"module": "commonjs"}' mocha --exit --timeout 10000 -r ts-node/register './src/tests/*.ts'

Setting up database...
Listening on port 5000

Notes Positive Test Cases
POST /
Successfully connected to DB
verbose: Note created {"note_id": "6144cdcf17024daea00d608f", "timestamp": "2021-09-17T17:17:54.819Z"}
http: HTTP POST /api/note 200 3676ms {"metadata": {"timestamp": "2021-09-17T17:17:54.825Z"}}
  ✓ Should create a note (3705ms)
  GET /
verbose: Notes retrieved {"notes_count": 2005, "timestamp": "2021-09-17T17:17:56.384Z"}
http: HTTP GET /api/note/ 200 1557ms {"metadata": {"timestamp": "2021-09-17T17:17:56.396Z"}}
  ✓ Should fetch all notes (1566ms)
verbose: Note retrieved {"note_id": "6144cdcf17024daea00d608f", "timestamp": "2021-09-17T17:17:56.634Z"}
http: HTTP GET /api/note/6144cdcf17024daea00d608f 200 229ms {"metadata": {"timestamp": "2021-09-17T17:17:56.641Z"}}
  ✓ Should fetch single note that was previously created (243ms)
  PUT /
verbose: Note updated {"note_id": "6144cdcf17024daea00d608f", "timestamp": "2021-09-17T17:17:57.331Z"}
http: HTTP PUT /api/note 200 689ms {"metadata": {"timestamp": "2021-09-17T17:17:57.337Z"}}
  ✓ Should update note that was previously created (693ms)
  DELETE /
verbose: Note soft-deleted {"note_id": "6144cdcf17024daea00d608f", "timestamp": "2021-09-17T17:17:57.799Z"}
http: HTTP DELETE /api/note/6144cdcf17024daea00d608f 200 457ms {"metadata": {"timestamp": "2021-09-17T17:17:57.800Z"}}
  ✓ Should soft delete note that was previously created (462ms)
verbose: Note hard-deleted {"note_id": "6144cdcf17024daea00d608f", "timestamp": "2021-09-17T17:17:58.266Z"}
http: HTTP DELETE /api/note/hard-delete/6144cdcf17024daea00d608f 200 457ms {"metadata": {"timestamp": "2021-09-17T17:17:58.261Z"}}
  ✓ Should hard delete note that was previously created (459ms)

Notes Negative Test Cases
POST /
http: HTTP POST /api/note 422 1ms {"metadata": {"timestamp": "2021-09-17T17:17:58.266Z"}}
  ✓ Should create a note
  GET /
http: HTTP GET /api/note/123 422 1ms {"metadata": {"timestamp": "2021-09-17T17:17:58.271Z"}}
  ✓ Should not fetch a note due to invalid ID
error: Note 613f4186abe8ac519b619877 not found. {"timestamp": "2021-09-17T17:17:58.502Z"}
http: HTTP GET /api/note/613f4186abe8ac519b619877 404 227ms {"metadata": {"timestamp": "2021-09-17T17:17:58.503Z"}}
  ✓ Should not fetch a note due to object not found (230ms)
  PUT /
http: HTTP PUT /api/note 422 1ms {"metadata": {"timestamp": "2021-09-17T17:17:58.507Z"}}
  ✓ Should not update any note due to invalid ID
error: Note 613f4186abe8ac519b619877 not found. {"timestamp": "2021-09-17T17:17:58.737Z"}
http: HTTP PUT /api/note 404 226ms {"metadata": {"timestamp": "2021-09-17T17:17:58.738Z"}}
  ✓ Should not update any note due to object not found (230ms)
  DELETE /
http: HTTP DELETE /api/note/123 422 0ms {"metadata": {"timestamp": "2021-09-17T17:17:58.749Z"}}
  ✓ Should not soft delete note due to invalid ID
error: Note 613f4186abe8ac519b619877 not found. {"timestamp": "2021-09-17T17:17:58.987Z"}
http: HTTP DELETE /api/note/613f4186abe8ac519b619877 404 227ms {"metadata": {"timestamp": "2021-09-17T17:17:58.988Z"}}
  ✓ Should not soft delete note due to object not found (232ms)
http: HTTP DELETE /api/note/hard-delete/123 422 1ms {"metadata": {"timestamp": "2021-09-17T17:17:58.994Z"}}
  ✓ Should not hard delete note due to invalid ID
error: Note 613f4186abe8ac519b619877 not found. {"timestamp": "2021-09-17T17:17:59.230Z"}
http: HTTP DELETE /api/note/hard-delete/613f4186abe8ac519b619877 404 226ms {"metadata": {"timestamp": "2021-09-17T17:17:59.231Z"}}
  ✓ Should not hard delete note due to invalid ID (231ms)

15 passing (8s)

The command "npm test" exited with 0.
store build cache
cache:2

Done. Your build exited with 0.
```

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.

```
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.

```
$ npm install -g serverless
$ openssl aes-256-cbc -K $encrypted_3c84dc6bbe_key -iv $encrypted_3c84dc6bbe_iv -in .env.enc -out .env -d
$ openssl aes-256-cbc -K $encrypted_4e8c5512ae30_key -iv $encrypted_4e8c5512ae30_iv -in serverless-key.json.enc -out serverless-key.json -d
$ npm run deploy

> [secure]1@1.0.0 deploy
> sls deploy

Serverless: DOTENV: Loading environment variables from .env:
Serverless:   - MONGO_USERNAME
Serverless:   - MONGO_PASSWORD
Serverless:   - MONGO_DB
Serverless: Configuration warning at 'provider.region': should be equal to one of the allowed values [us-central1, us-east1, us-east4, europe-west1, europe-west2, asia-east2, asia-northeast1, asia-northeast2, us-west2, us-west3, us-west4, northamerica-northeast1, southamerica-east1, europe-west3, europe-west6, australia-southeast1, asia-south1, asia-southeast2, asia-northeast3]
Serverless:
Serverless: Learn more about configuration validation here: http://sls.io/configuration-validation
Serverless:
Serverless: Compiling with Typescript...
Serverless: Using local tsconfig.json
Serverless: Typescript compiled.
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Compiling function "app"...
Serverless: Uploading artifacts...
Serverless: Artifacts successfully uploaded...
Serverless: Updating deployment...
Serverless: Checking deployment update progress...
................................................................
................................................................
................................................................
Serverless: Done...
Service Information
service: [secure]
project: [secure]-325509
stage: dev
region: asia-southeast1

Deployed functions
app
  https://asia-southeast1-[secure]-325509.cloudfunctions.net/[secure]-dev-app

Serverless: Removing old artifacts...
Serverless: Deprecation warnings:

Starting with next major, Serverless will throw on configuration errors by default. Adapt to this behavior now by adding "configValidationMode: error" to service configuration
More Info: https://www.serverless.com/framework/docs/deprecations/#CONFIG_VALIDATION_MODE_DEFAULT

Support for "package.include" and "package.exclude" will be removed with next major release. Please use "package.patterns" instead
More Info: https://www.serverless.com/framework/docs/deprecations/#NEW_PACKAGE_PATTERNS

The command "npm run deploy" exited with 0.
store build cache
cache.2

Done. Your build exited with 0.
```

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

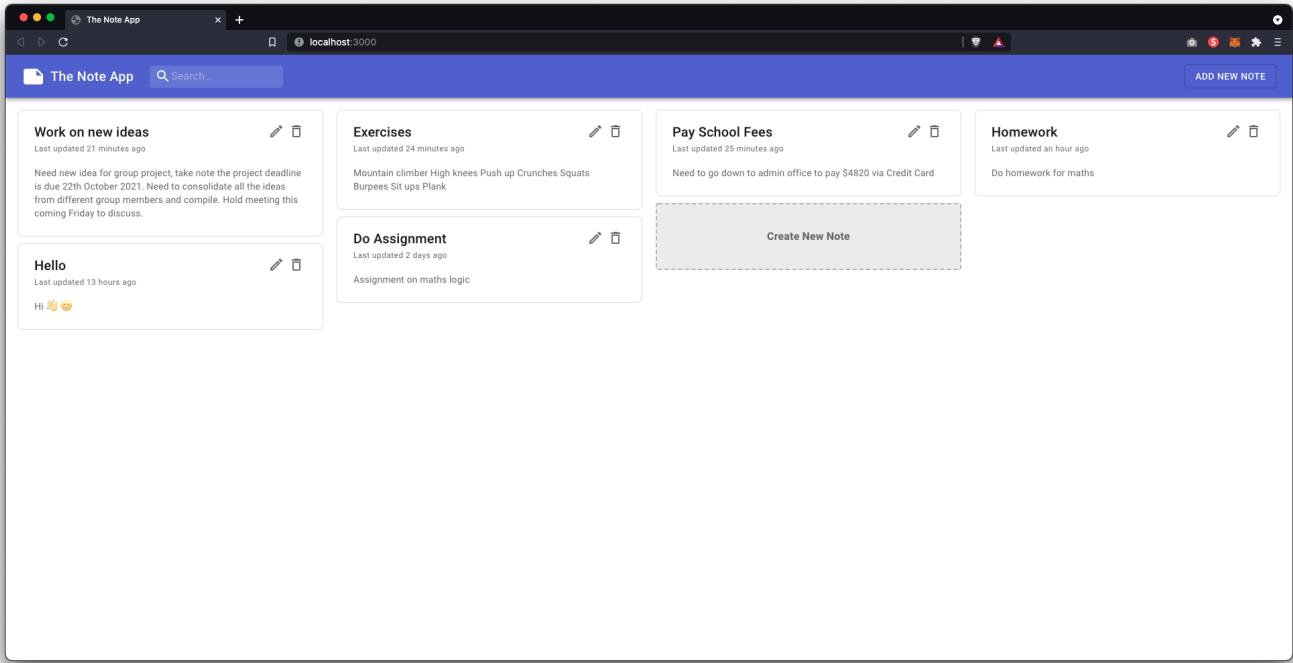
```
npm run dev
```

Endpoint

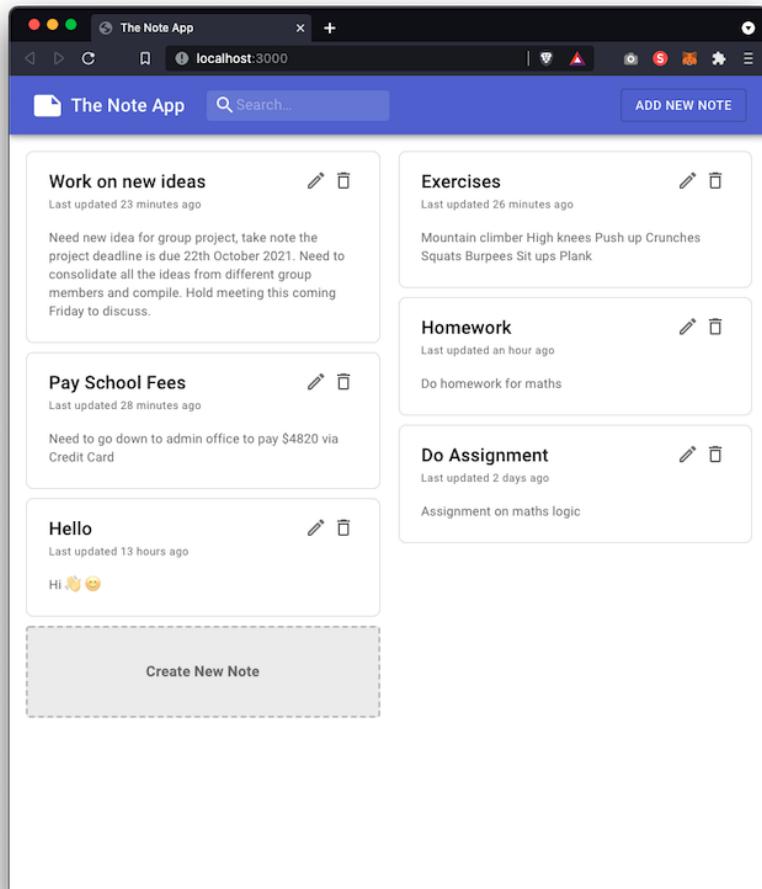
- Localhost: <http://localhost:3000>
- Deployed Endpoint: <https://cs3219-otot-task-b-325509.firebaseioapp.com/>

Demonstration

Desktop View



Tablet / Smaller Screen View



Mobile View

ADD

Work on new ideas edit remove

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 edit remove

Last updated 27 minutes ago

Mountain climber High knees Push up Crunches Squats Burpees Sit ups Plank

Pay School Fees edit remove

Last updated 28 minutes ago

Need to go down to admin office to pay \$4820 via Credit Card

Homework edit remove

Last updated an hour ago

Do homework for maths

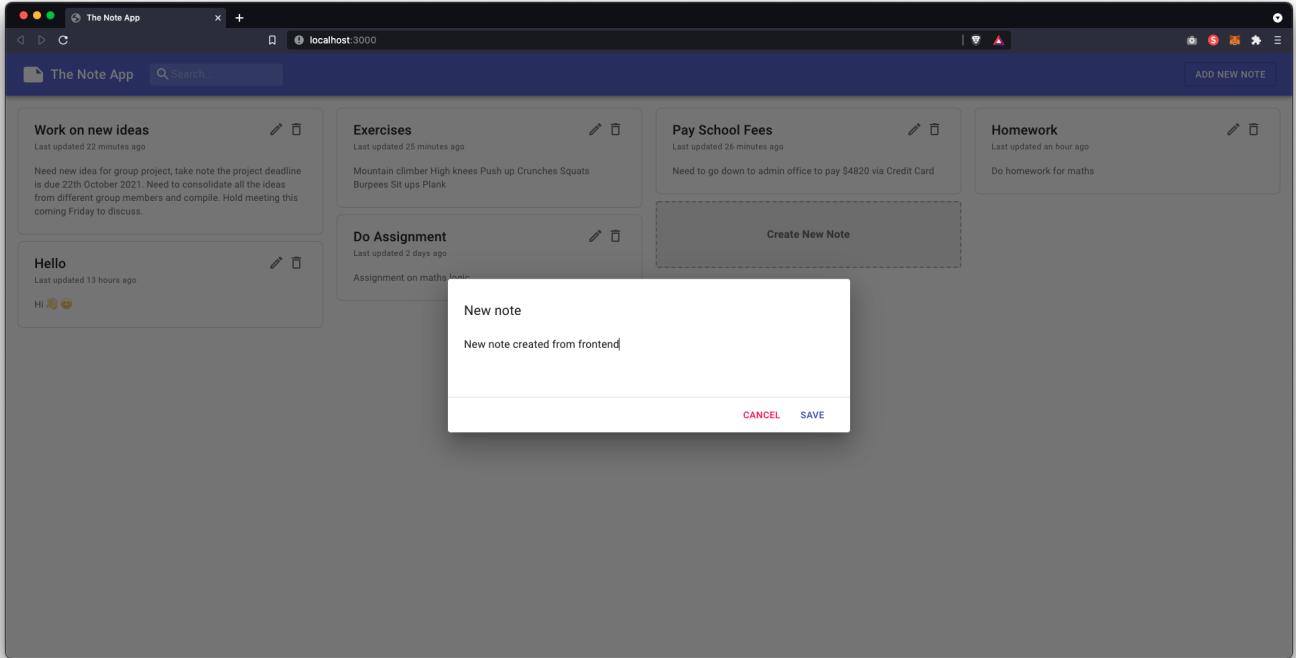
Hello edit remove

Last updated 13 hours ago

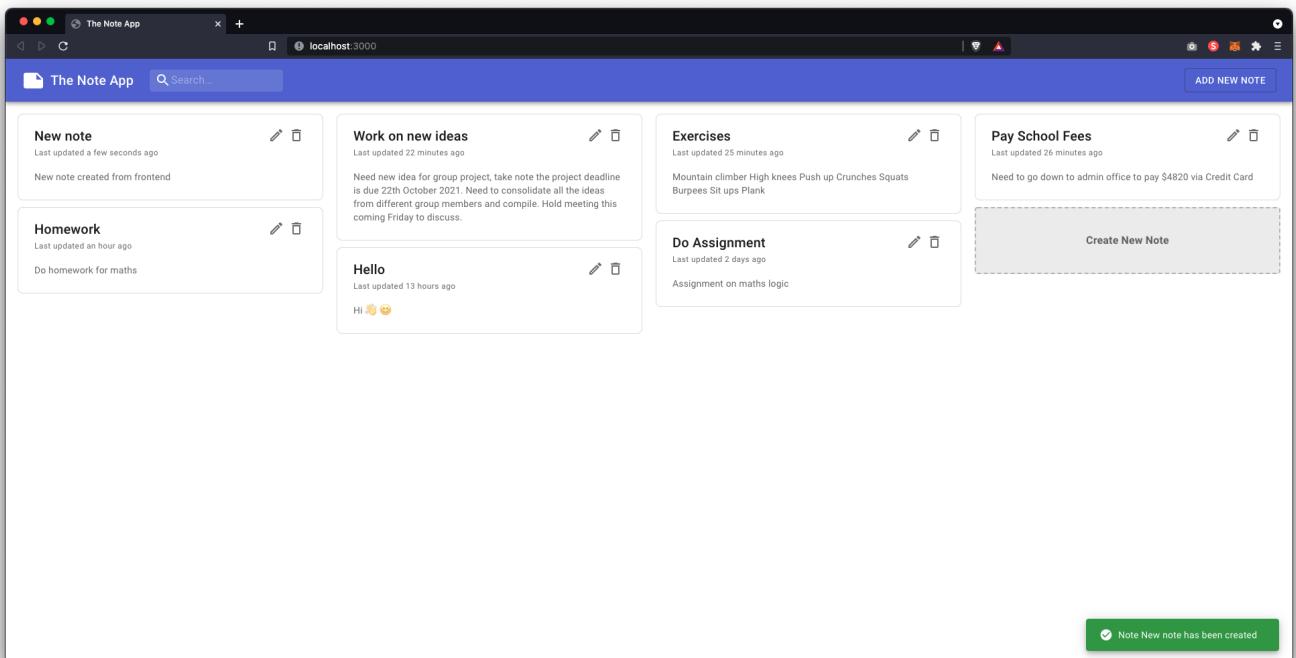
Create

- 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

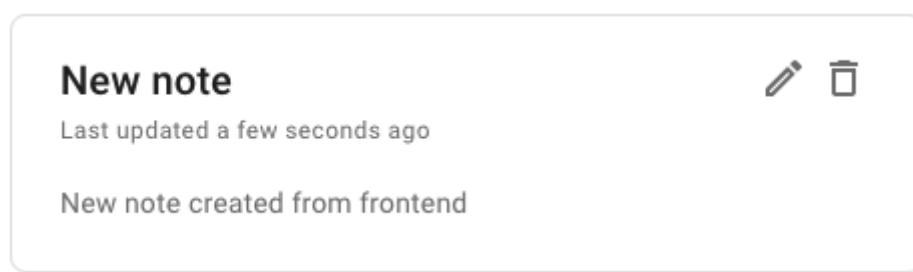


- After the note is created, a notification will appear on the bottom right of the screen, and the new note is displayed as the first item in the masonry grid.

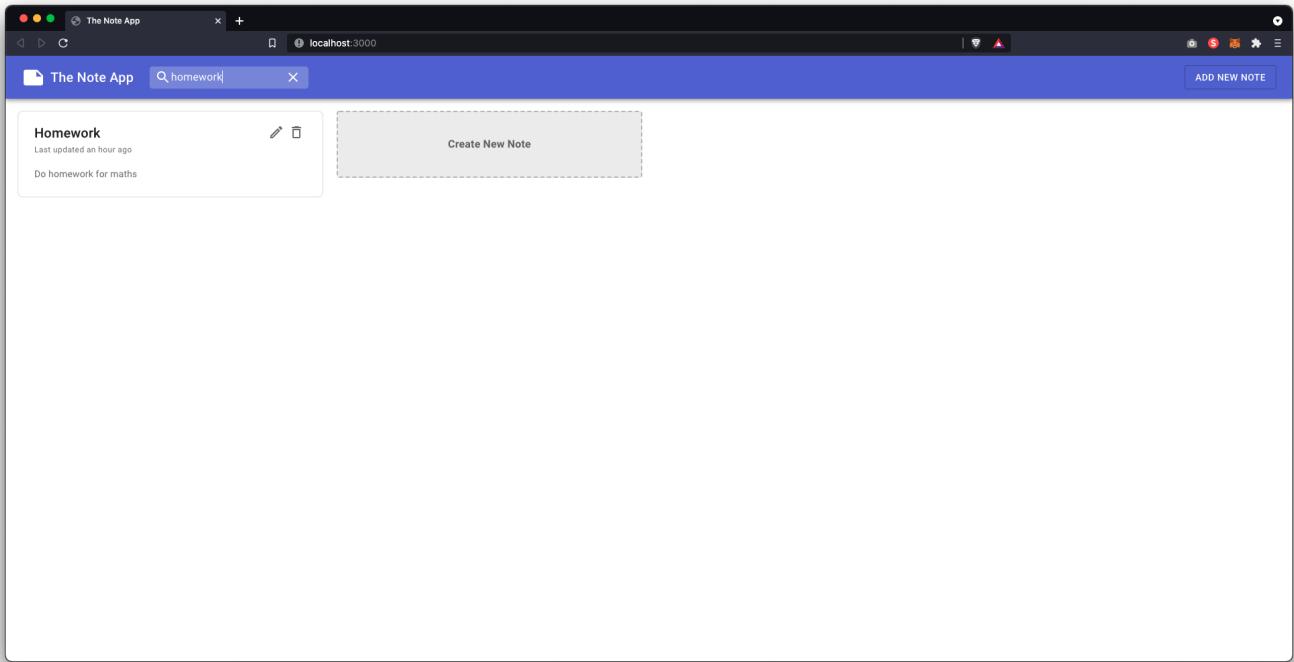


Retrieve

- 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.

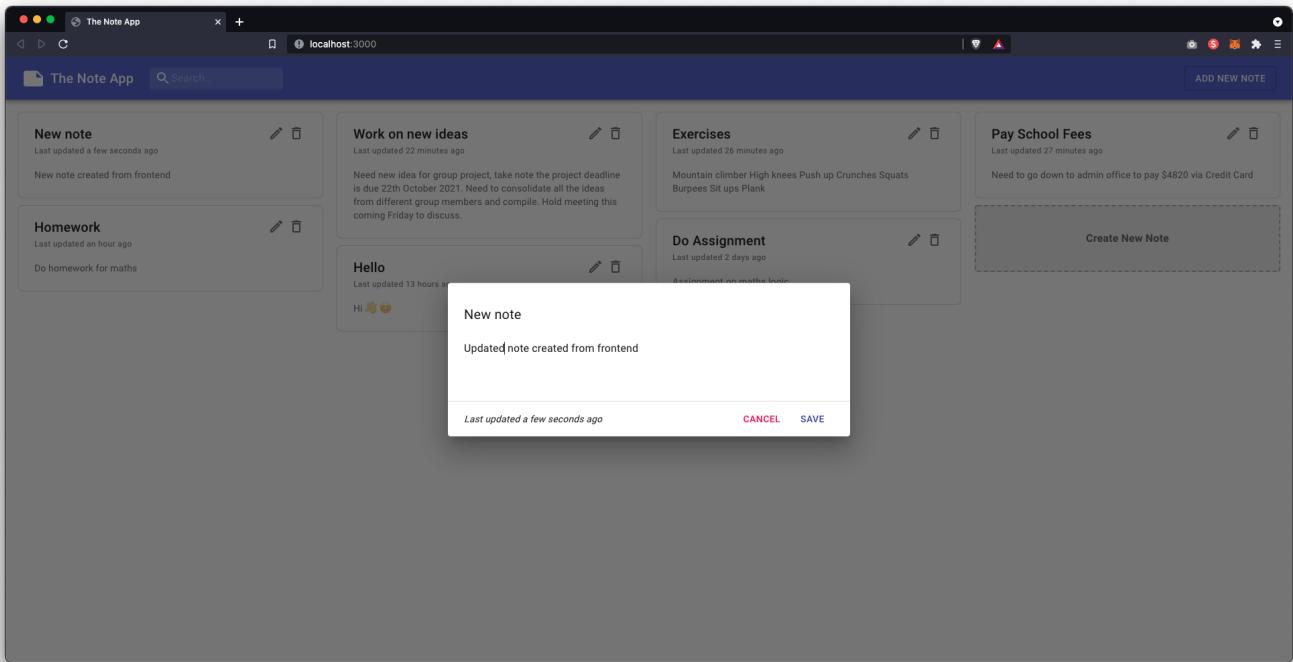


- You can perform searching (server-side filtering) by typing to the search box in the app bar.

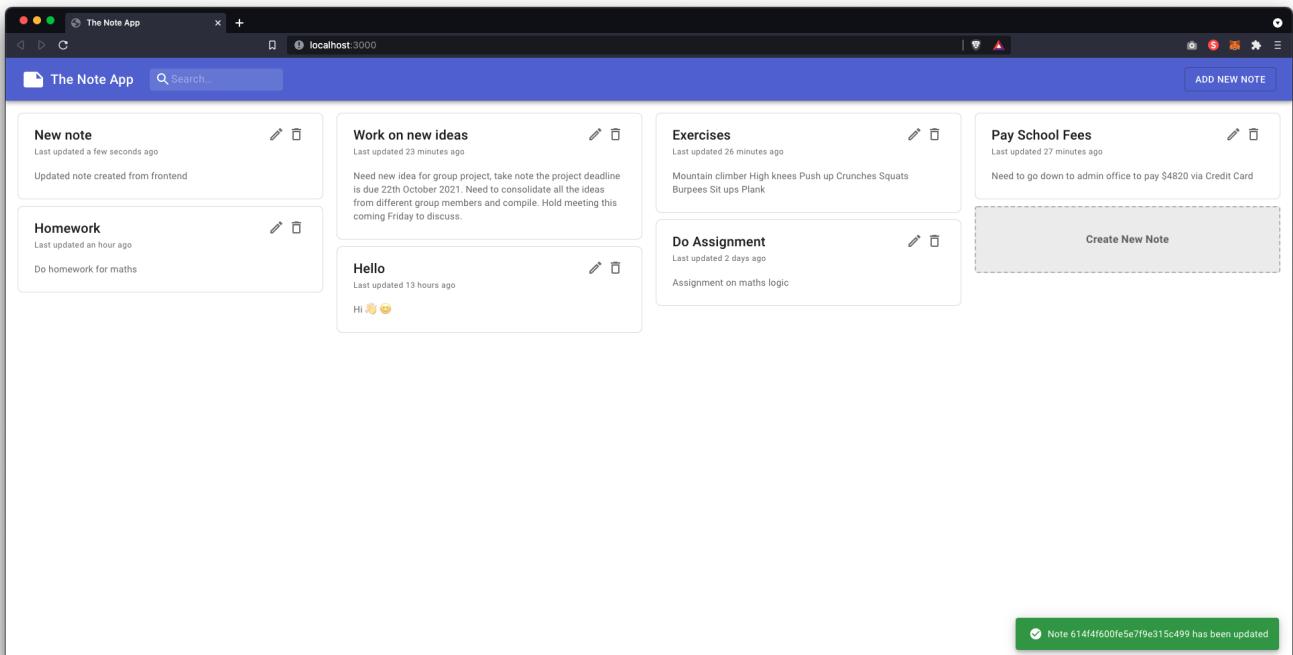


Update

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

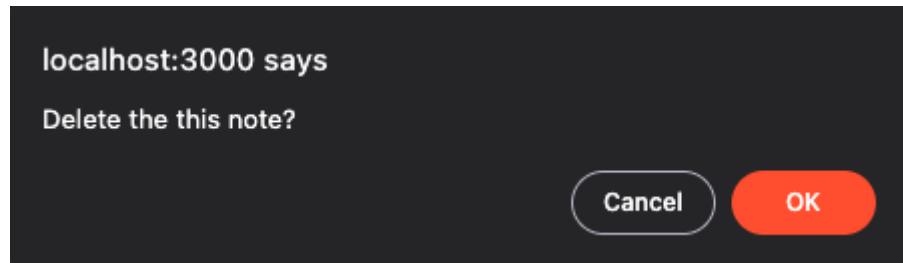


- After the note is created, a notification will appear on the bottom right of the screen, and the updated note is displayed as the first item in the masonry grid.



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.

The Note App

ADD NEW NOTE

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.

Hello

Last updated 13 hours ago

Hi 😊 😊

Exercises

Last updated 26 minutes ago

Mountain climber High knees Push up Crunches Squats Burpees Sit ups Plank

Do Assignment

Last updated 2 days ago

Assignment on maths logic

Pay School Fees

Last updated 27 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

Create New Note

Note 614f4f600fe5e7f9e315c499 has been deleted

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>