IT314
Software Engineering

Group 5
Hostel Management System

Unit Testing Document

Instructor: Dr. Saurabh Tiwari and Dr. Manish Khare

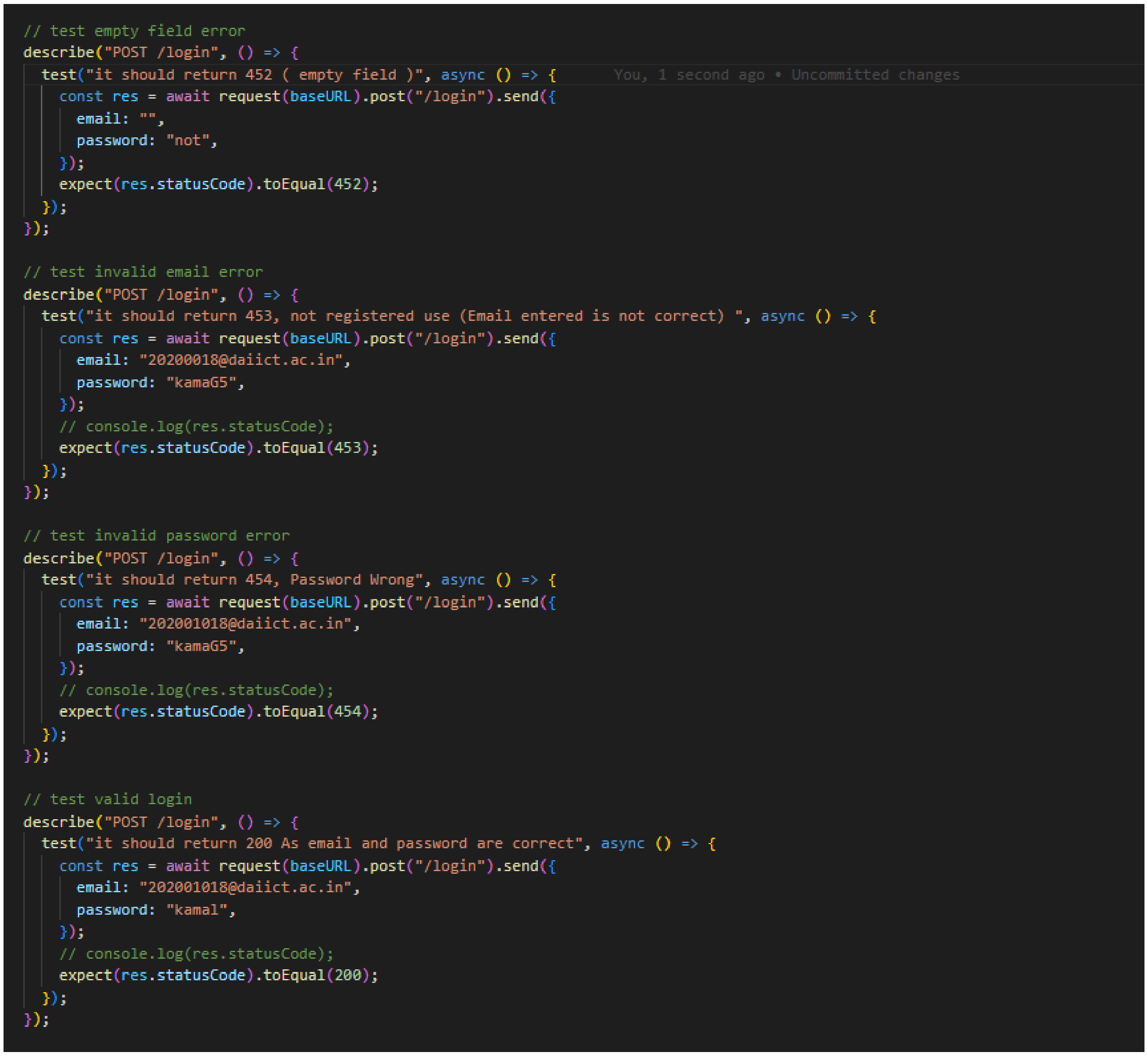
Date: 26/04/2023

- We have used Used Jest Framework to do Unit testing for different API’s in our Program.
- During our work we have used Postman to check if POST/GET/PUT/DELTE request are working fine in backend or not.

- In This document we have provided
    - API to be tested
    - Test File using Jest
    - Test output
    - Explanation of Each test case

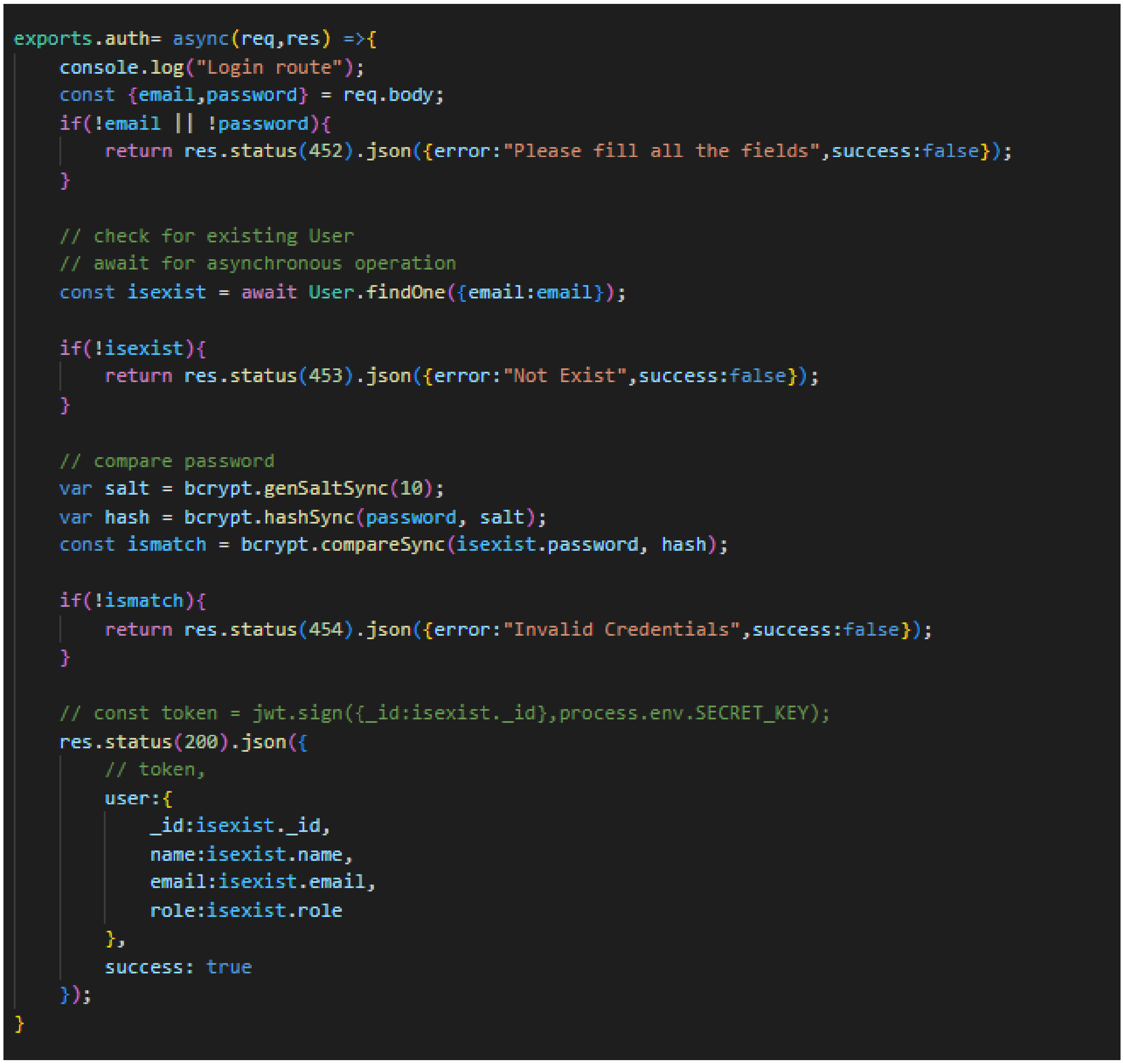
1. Authentication

Test code:



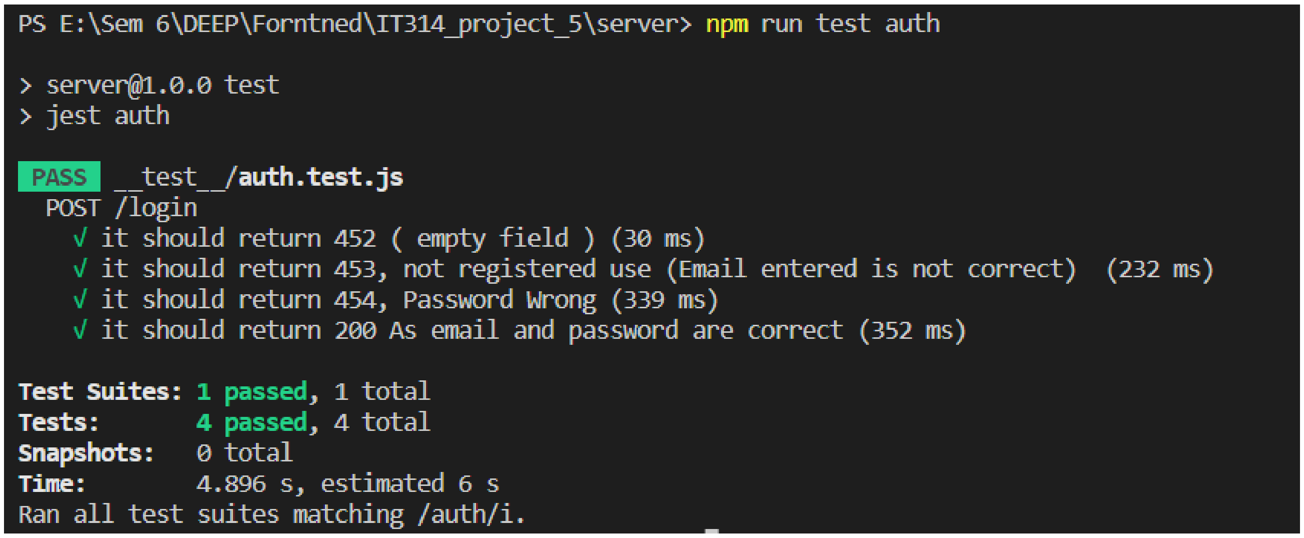
```
// test empty field error
describe("POST /login", () => {
  test("it should return 452 ( empty field )", async () => {        You, 1 second ago • Uncommitted changes
    const res = await request(baseURL).post("/login").send({
      email: "",
      password: "not",
    });
    expect(res.statusCode).toEqual(452);
  });
});

// test invalid email error
describe("POST /login", () => {
  test("it should return 453, not registered use (Email entered is not correct) ", async () => {
    const res = await request(baseURL).post("/login").send({
      email: "20200018@daiict.ac.in",
      password: "kamaG5",
    });
    // console.log(res.statusCode);
    expect(res.statusCode).toEqual(453);
  });
});

// test invalid password error
describe("POST /login", () => {
  test("it should return 454, Password Wrong", async () => {
    const res = await request(baseURL).post("/login").send({
      email: "202001018@daiict.ac.in",
      password: "kamaG5",
    });
    // console.log(res.statusCode);
    expect(res.statusCode).toEqual(454);
  });
});

// test valid login
describe("POST /login", () => {
  test("it should return 200 As email and password are correct", async () => {
    const res = await request(baseURL).post("/login").send({
      email: "202001018@daiict.ac.in",
      password: "kamal",
    });
    // console.log(res.statusCode);
    expect(res.statusCode).toEqual(200);
  });
});
```

API to be Tested:



```
exports.auth= async(req,res) =>{
    console.log("Login route");
    const {email,password} = req.body;
    if(!email || !password){
        return res.status(452).json({error:"Please fill all the fields",success:false});
    }

    // check for existing User
    // await for asynchronous operation
    const isexist = await User.findOne({email:email});

    if(!isexist){
        return res.status(453).json({error:"Not Exist",success:false});
    }

    // compare password
    var salt = bcrypt.genSaltSync(10);
    var hash = bcrypt.hashSync(password, salt);
    const ismatch = bcrypt.compareSync(isexist.password, hash);

    if(!ismatch){
        return res.status(454).json({error:"Invalid Credentials",success:false});
    }

    // const token = jwt.sign({_id:isexist._id},process.env.SECRET_KEY);
    res.status(200).json({
        // token,
        user:{
            _id:isexist._id,
            name:isexist.name,
            email:isexist.email,
            role:isexist.role
        },
        success: true
    });
}
```

Test Results:



```
PS E:\Sem 6\DEEP\Forntned\IT314_project_5\server> npm run test auth

> server@1.0.0 test
> jest auth

 PASS  __test__/auth.test.js
  POST /login
    √ it should return 452 ( empty field ) (30 ms)
    √ it should return 453, not registered use (Email entered is not correct)  (232 ms)
    √ it should return 454, Password Wrong (339 ms)
    √ it should return 200 As email and password are correct (352 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        4.896 s, estimated 6 s
Ran all test suites matching /auth/i.
```

Explanation :

1. Test case: Empty Field Error

This test case checks if the API returns a status code of 452 when the email field is empty while attempting to log in.

2. Test case: Invalid Email Error

This test case checks if the API returns a status code of 453 when the email entered is not registered or incorrect while attempting to log in.

3. Test case: Invalid Password Error

This test case checks, if the API returns a status code of 454 when the password entered is incorrect while attempting to log in.

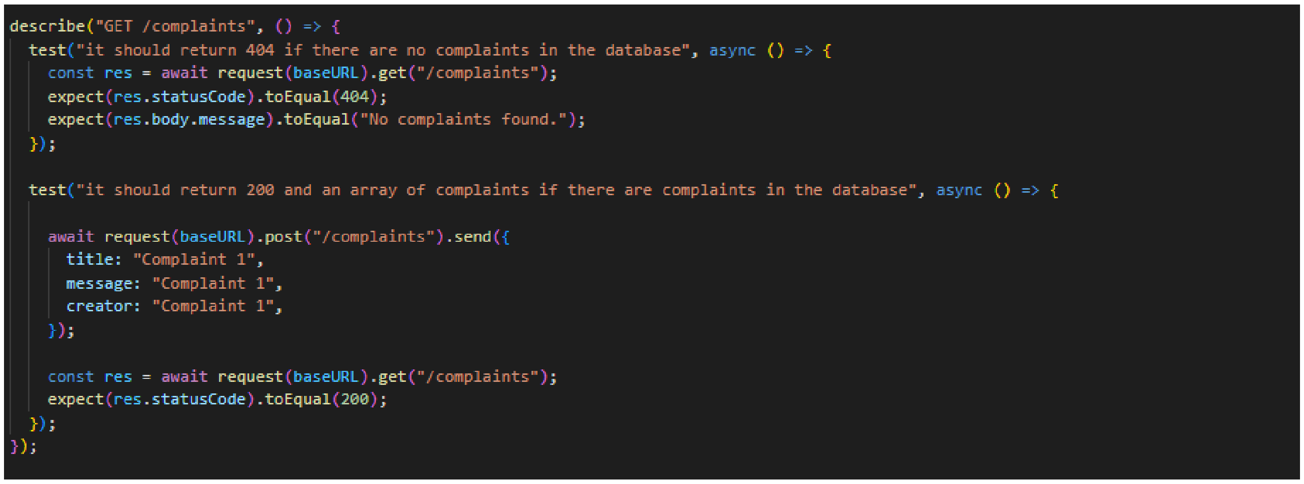
4. Test case: Valid Login

This test case checks if a valid login attempt returns a status code of 200, which indicates a successful login.

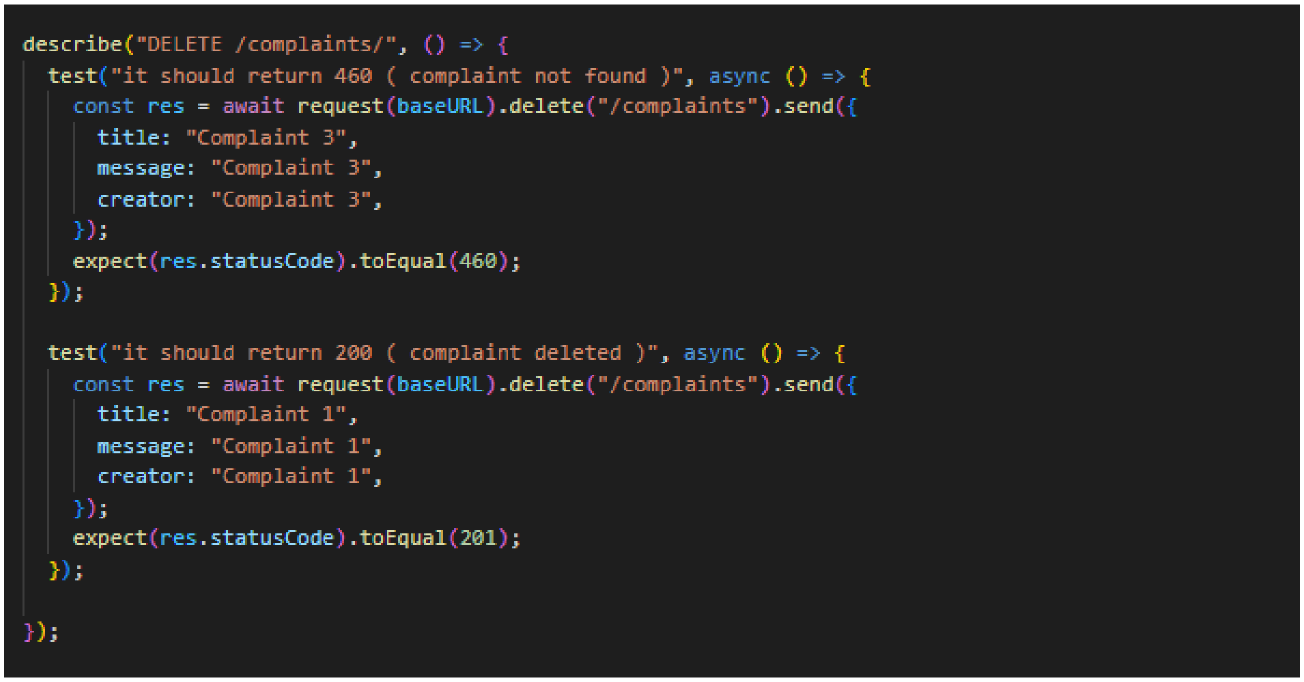
Time: 4.896 s, estimated 6 s

1. Complaint

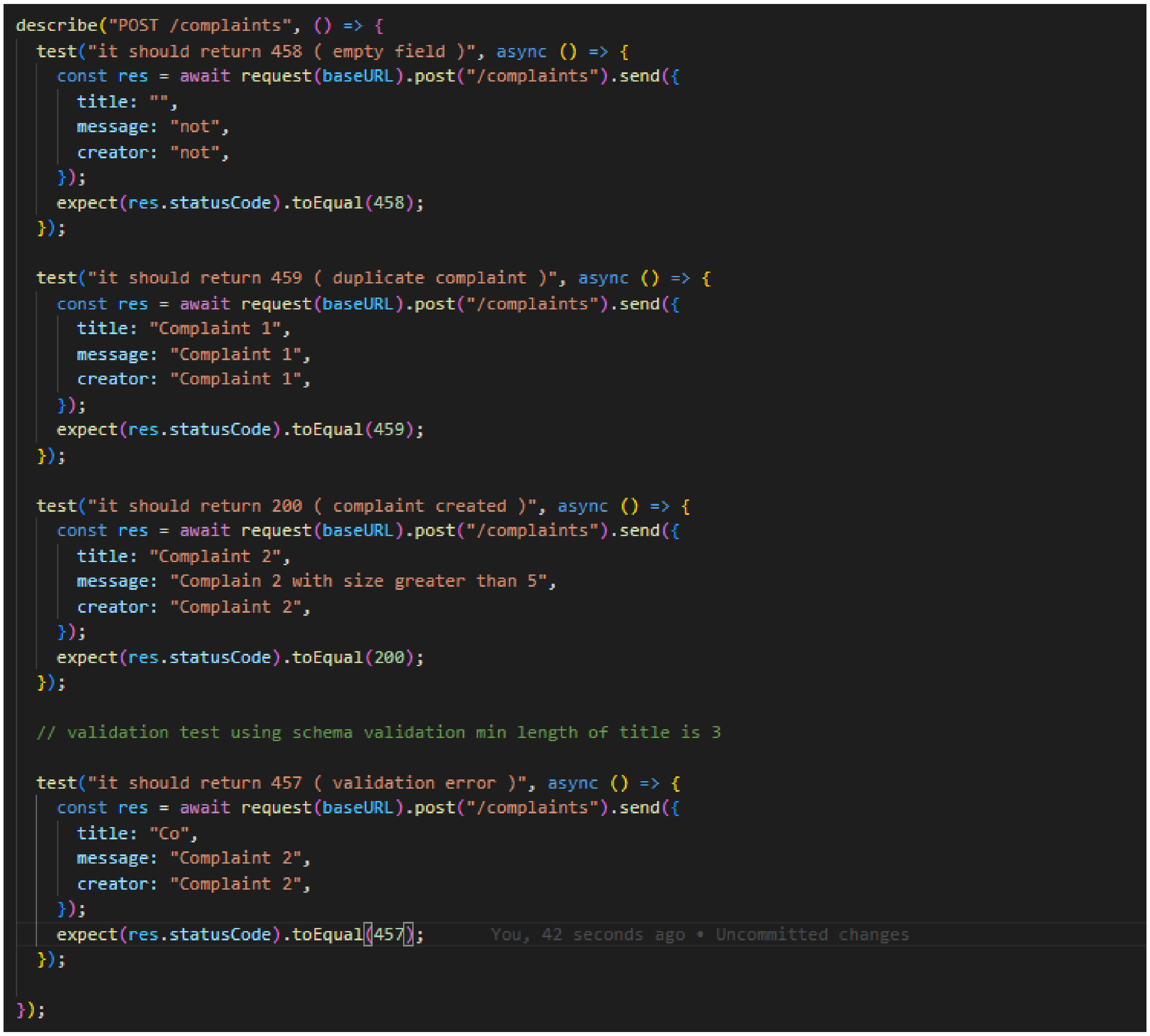
Test Codes:



```
describe("GET /complaints", () => {
  test("it should return 404 if there are no complaints in the database", async () => {
    const res = await request(baseURL).get("/complaints");
    expect(res.statusCode).toEqual(404);
    expect(res.body.message).toEqual("No complaints found.");
  });

  test("it should return 200 and an array of complaints if there are complaints in the database", async () => {

    await request(baseURL).post("/complaints").send({
      title: "Complaint 1",
      message: "Complaint 1",
      creator: "Complaint 1",
    });

    const res = await request(baseURL).get("/complaints");
    expect(res.statusCode).toEqual(200);
  });
});
```

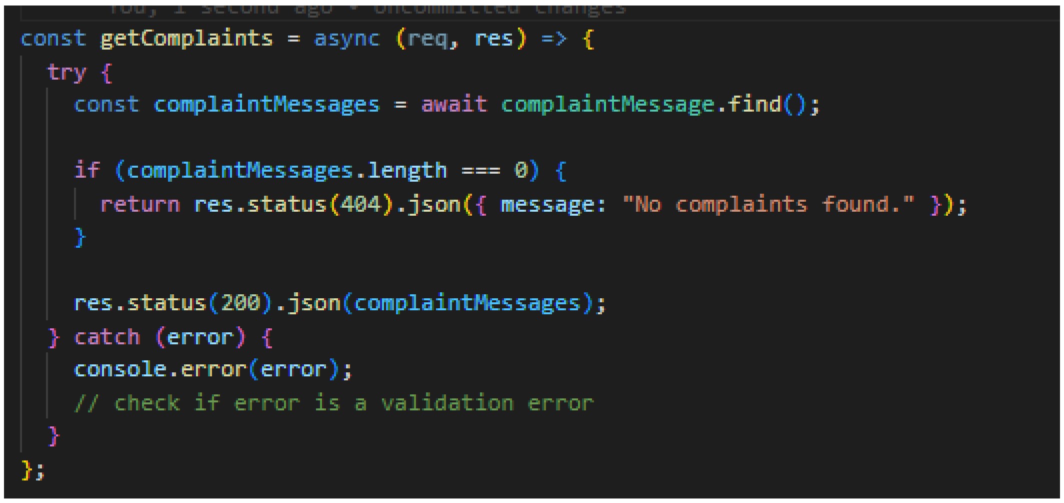


```
describe("DELETE /complaints/", () => {
  test("it should return 460 ( complaint not found )", async () => {
    const res = await request(baseURL).delete("/complaints").send({
      title: "Complaint 3",
      message: "Complaint 3",
      creator: "Complaint 3",
    });
    expect(res.statusCode).toEqual(460);
  });

  test("it should return 200 ( complaint deleted )", async () => {
    const res = await request(baseURL).delete("/complaints").send({
      title: "Complaint 1",
      message: "Complaint 1",
      creator: "Complaint 1",
    });
    expect(res.statusCode).toEqual(201);
  });

});
```

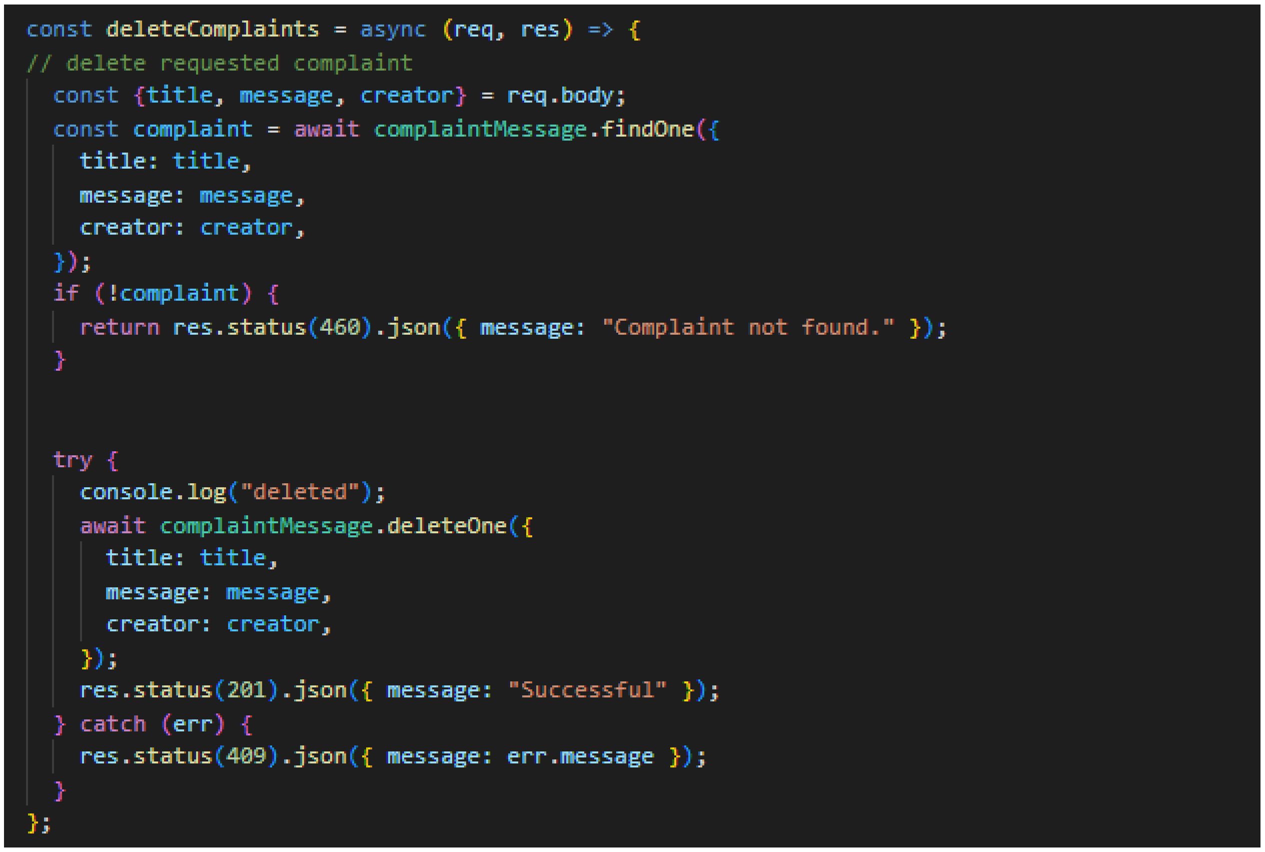


```javascript
describe("POST /complaints", () => {
  test("it should return 458 ( empty field )", async () => {
    const res = await request(baseURL).post("/complaints").send({
      title: "",
      message: "not",
      creator: "not",
    });
    expect(res.statusCode).toEqual(458);
  });

  test("it should return 459 ( duplicate complaint )", async () => {
    const res = await request(baseURL).post("/complaints").send({
      title: "Complaint 1",
      message: "Complaint 1",
      creator: "Complaint 1",
    });
    expect(res.statusCode).toEqual(459);
  });

  test("it should return 200 ( complaint created )", async () => {
    const res = await request(baseURL).post("/complaints").send({
      title: "Complaint 2",
      message: "Complain 2 with size greater than 5",
      creator: "Complaint 2",
    });
    expect(res.statusCode).toEqual(200);
  });

  // validation test using schema validation min length of title is 3

  test("it should return 457 ( validation error )", async () => {
    const res = await request(baseURL).post("/complaints").send({
      title: "Co",
      message: "Complaint 2",
      creator: "Complaint 2",
    });
    expect(res.statusCode).toEqual(457);      You, 42 seconds ago • Uncommitted changes
  });

});
```

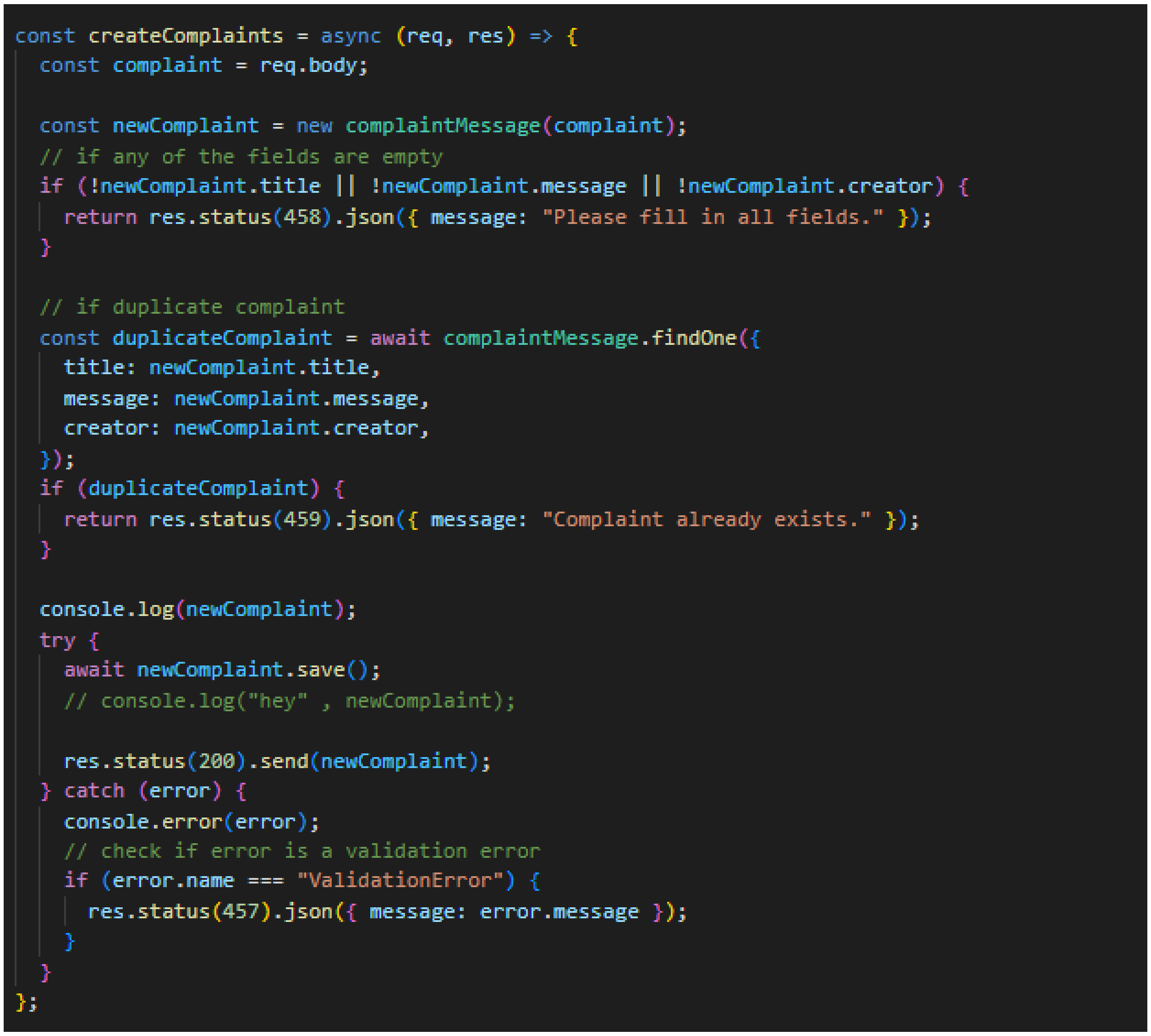
API To be Tested:



```
const getComplaints = async (req, res) => {
  try {
    const complaintMessages = await complaintMessage.find();

    if (complaintMessages.length === 0) {
      return res.status(404).json({ message: "No complaints found." });
    }

    res.status(200).json(complaintMessages);
  } catch (error) {
    console.error(error);
    // check if error is a validation error
  }
};
```

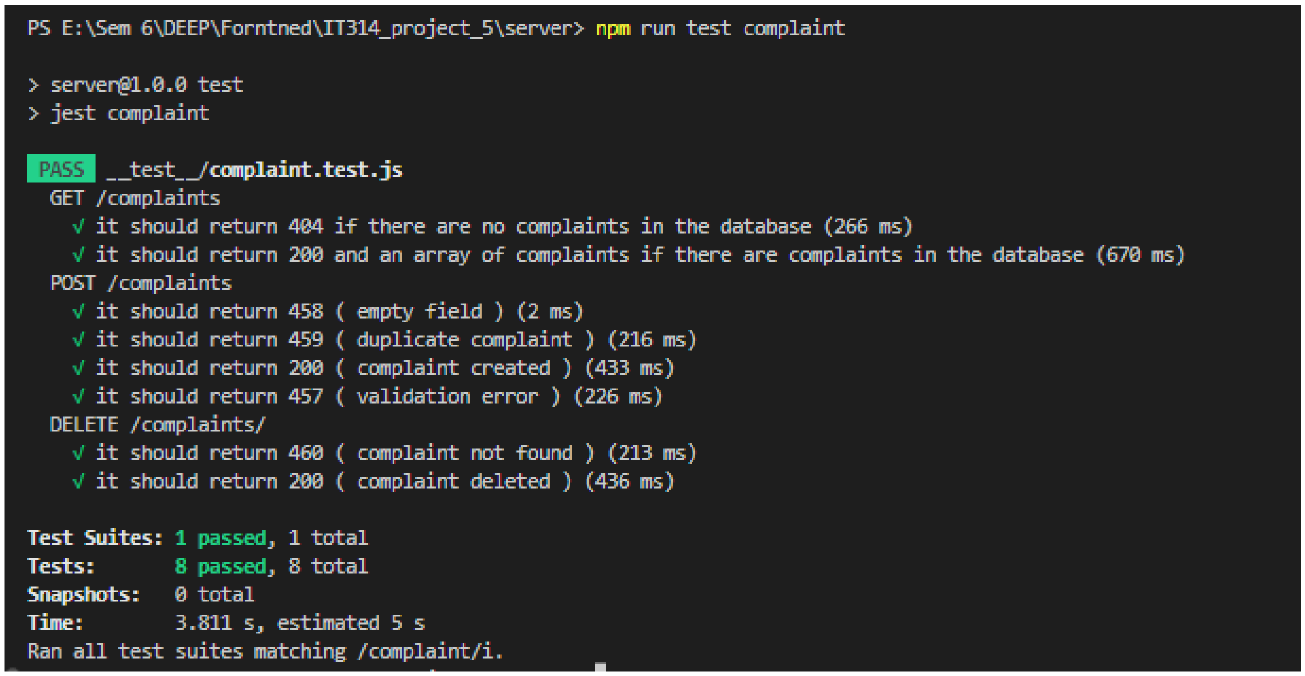


```
const deleteComplaints = async (req, res) => {
// delete requested complaint
  const {title, message, creator} = req.body;
  const complaint = await complaintMessage.findOne({
    title: title,
    message: message,
    creator: creator,
  });
  if (!complaint) {
    return res.status(460).json({ message: "Complaint not found." });
  }


  try {
    console.log("deleted");
    await complaintMessage.deleteOne({
      title: title,
      message: message,
      creator: creator,
    });
    res.status(201).json({ message: "Successful" });
  } catch (err) {
    res.status(409).json({ message: err.message });
  }
};
```



```
const createComplaints = async (req, res) => {
  const complaint = req.body;

  const newComplaint = new complaintMessage(complaint);
  // if any of the fields are empty
  if (!newComplaint.title || !newComplaint.message || !newComplaint.creator) {
    return res.status(458).json({ message: "Please fill in all fields." });
  }

  // if duplicate complaint
  const duplicateComplaint = await complaintMessage.findOne({
    title: newComplaint.title,
    message: newComplaint.message,
    creator: newComplaint.creator,
  });
  if (duplicateComplaint) {
    return res.status(459).json({ message: "Complaint already exists." });
  }

  console.log(newComplaint);
  try {
    await newComplaint.save();
    // console.log("hey" , newComplaint);

    res.status(200).send(newComplaint);
  } catch (error) {
    console.error(error);
    // check if error is a validation error
    if (error.name === "ValidationError") {
      res.status(457).json({ message: error.message });
    }
  }
};
```

Test Results:



```
PS E:\Sem 6\DEEP\Forntned\IT314_project_5\server> npm run test complaint

> server@1.0.0 test
> jest complaint

 PASS  __test__/complaint.test.js
  GET /complaints
    √ it should return 404 if there are no complaints in the database (266 ms)
    √ it should return 200 and an array of complaints if there are complaints in the database (670 ms)
  POST /complaints
    √ it should return 458 ( empty field ) (2 ms)
    √ it should return 459 ( duplicate complaint ) (216 ms)
    √ it should return 200 ( complaint created ) (433 ms)
    √ it should return 457 ( validation error ) (226 ms)
  DELETE /complaints/
    √ it should return 460 ( complaint not found ) (213 ms)
    √ it should return 200 ( complaint deleted ) (436 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        3.811 s, estimated 5 s
Ran all test suites matching /complaint/i.
```

Explanation :

- **Get request** :

1. Test case: No Complaints found

This test case checks if the API returns a status code of 404 when No complaints found

2. Test case: Complaint successfully fetched

This test case checks if the API returns a status code of 200 when the complaint can be fetched from DB successfully.

- **Post Request:**

1. Test case: Empty Field

This test case checks, if the API returns a status code of 458 when user post Complaint with empty field (title, message, creator)

2. Test case: Duplicate Complaint

This test case checks if the API returns a status code of 459 when user post a complaint with fields that are already exist in the database.

3. Test Case: Complaint created Succesfully.

This test case checks if the API returns a status code of 200 when posted complaint is stored at DB successfully.

4. Test Case: Validation Error

This test case checks if the API returns a status code of 457 when posted rules is not validated by Schema or DB. In this case, we post a complaint with a title size< 3, which violates the schema condition that min length of a title is 3.

- ## Delete Request:

1. Test case: Complaint not found

This test case checks if the API returns a status code of 460 when the requested deleted complaint doesn’t exist in DB.

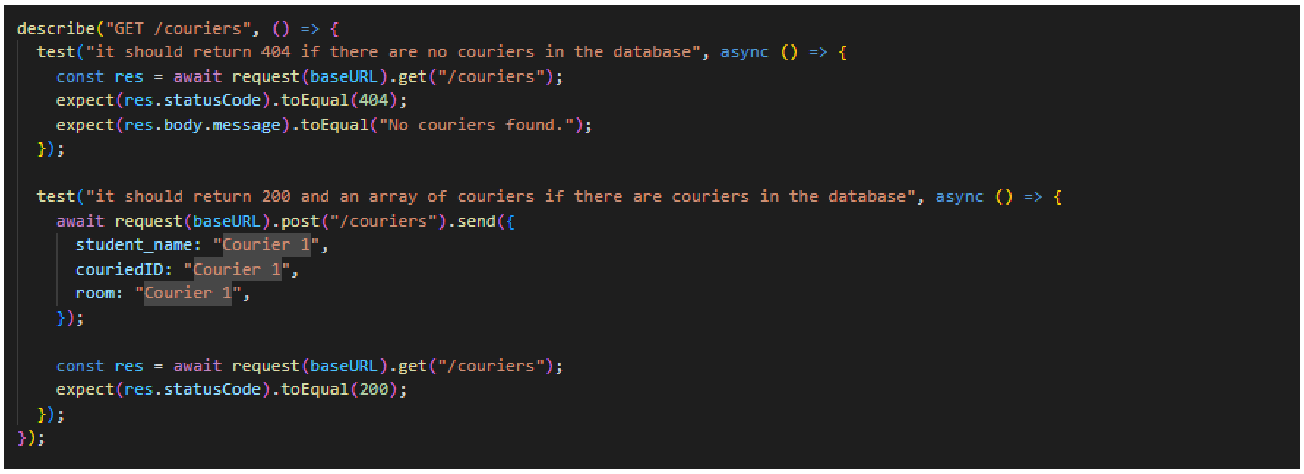
2. Test case: Complaint deleted successfully

This test case checks if the API returns a status code of 200 when the complaint can be deleted from DB successfully.

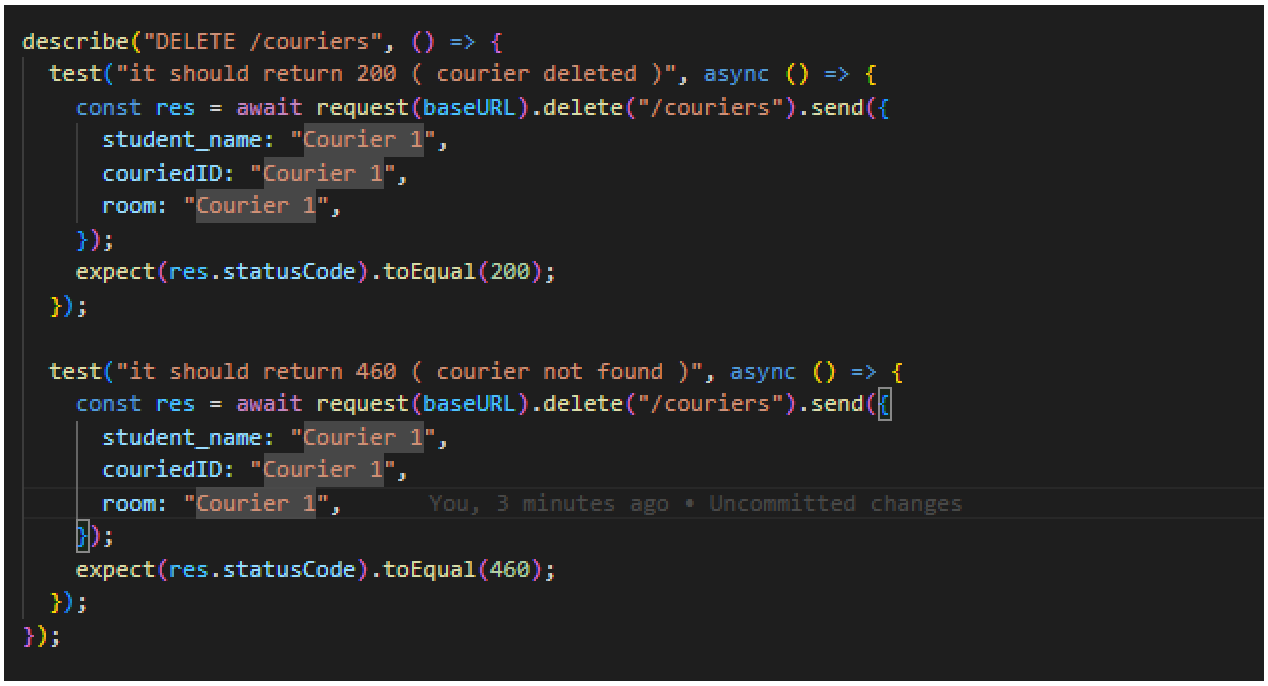
Time: 3.811 s, estimated 5 s

2. Couriours

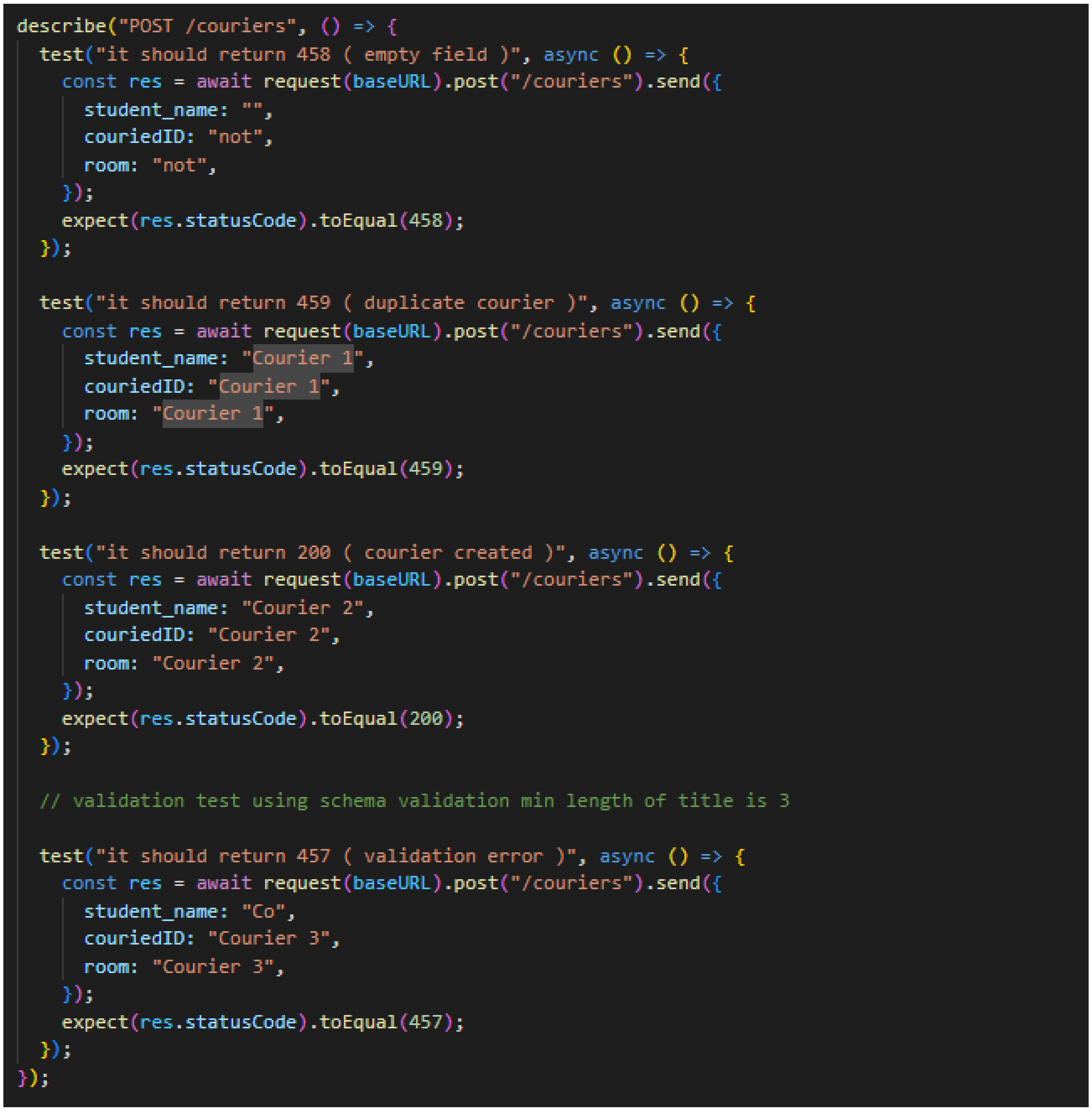
Test Codes:



```
describe("GET /couriers", () => {
  test("it should return 404 if there are no couriers in the database", async () => {
    const res = await request(baseURL).get("/couriers");
    expect(res.statusCode).toEqual(404);
    expect(res.body.message).toEqual("No couriers found.");
  });

  test("it should return 200 and an array of couriers if there are couriers in the database", async () => {
    await request(baseURL).post("/couriers").send({
      student_name: "Courier 1",
      couriedID: "Courier 1",
      room: "Courier 1",
    });

    const res = await request(baseURL).get("/couriers");
    expect(res.statusCode).toEqual(200);
  });
});
```

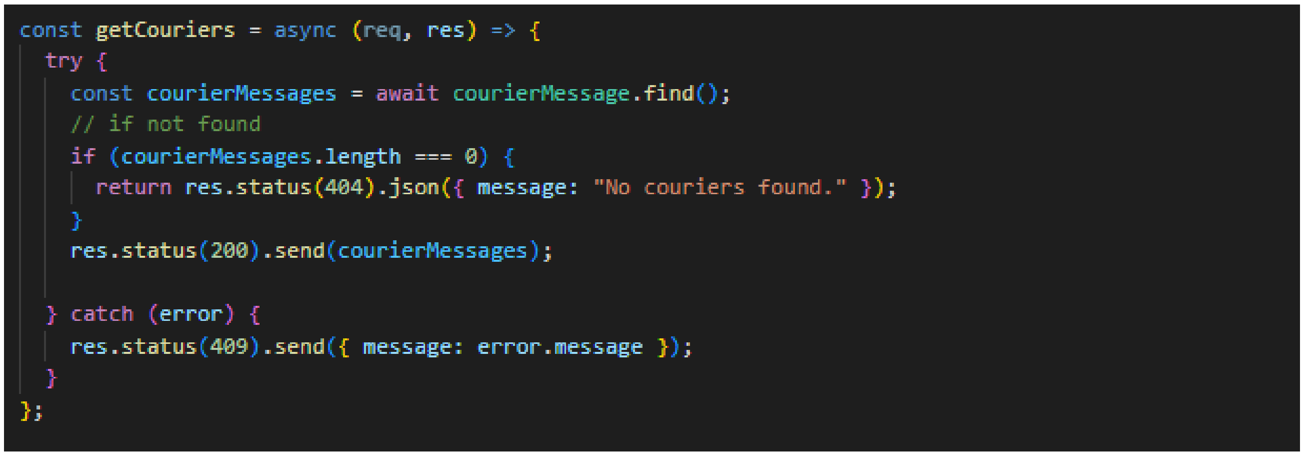


```
describe("DELETE /couriers", () => {
  test("it should return 200 ( courier deleted )", async () => {
    const res = await request(baseURL).delete("/couriers").send({
      student_name: "Courier 1",
      couriedID: "Courier 1",
      room: "Courier 1",
    });
    expect(res.statusCode).toEqual(200);
  });

  test("it should return 460 ( courier not found )", async () => {
    const res = await request(baseURL).delete("/couriers").send({
      student_name: "Courier 1",
      couriedID: "Courier 1",
      room: "Courier 1",        You, 3 minutes ago • Uncommitted changes
    });
    expect(res.statusCode).toEqual(460);
  });
});
```

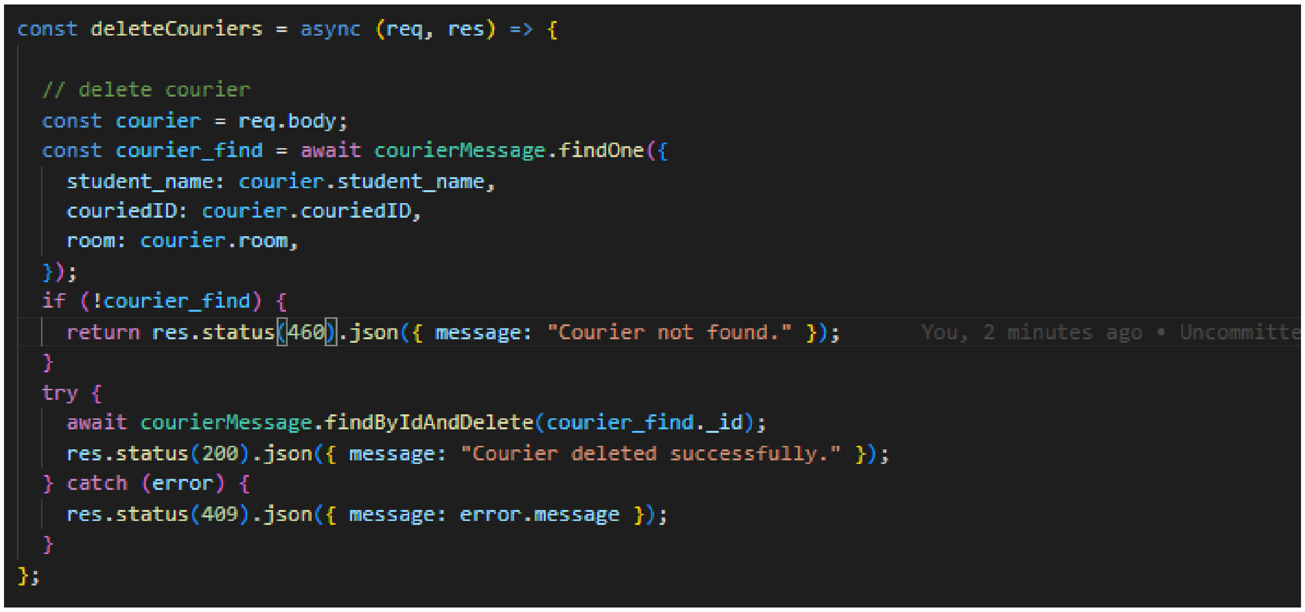


```javascript
describe("POST /couriers", () => {
  test("it should return 458 ( empty field )", async () => {
    const res = await request(baseURL).post("/couriers").send({
      student_name: "",
      couriedID: "not",
      room: "not",
    });
    expect(res.statusCode).toEqual(458);
  });

  test("it should return 459 ( duplicate courier )", async () => {
    const res = await request(baseURL).post("/couriers").send({
      student_name: "Courier 1",
      couriedID: "Courier 1",
      room: "Courier 1",
    });
    expect(res.statusCode).toEqual(459);
  });

  test("it should return 200 ( courier created )", async () => {
    const res = await request(baseURL).post("/couriers").send({
      student_name: "Courier 2",
      couriedID: "Courier 2",
      room: "Courier 2",
    });
    expect(res.statusCode).toEqual(200);
  });

  // validation test using schema validation min length of title is 3

  test("it should return 457 ( validation error )", async () => {
    const res = await request(baseURL).post("/couriers").send({
      student_name: "Co",
      couriedID: "Courier 3",
      room: "Courier 3",
    });
    expect(res.statusCode).toEqual(457);
  });
});
```

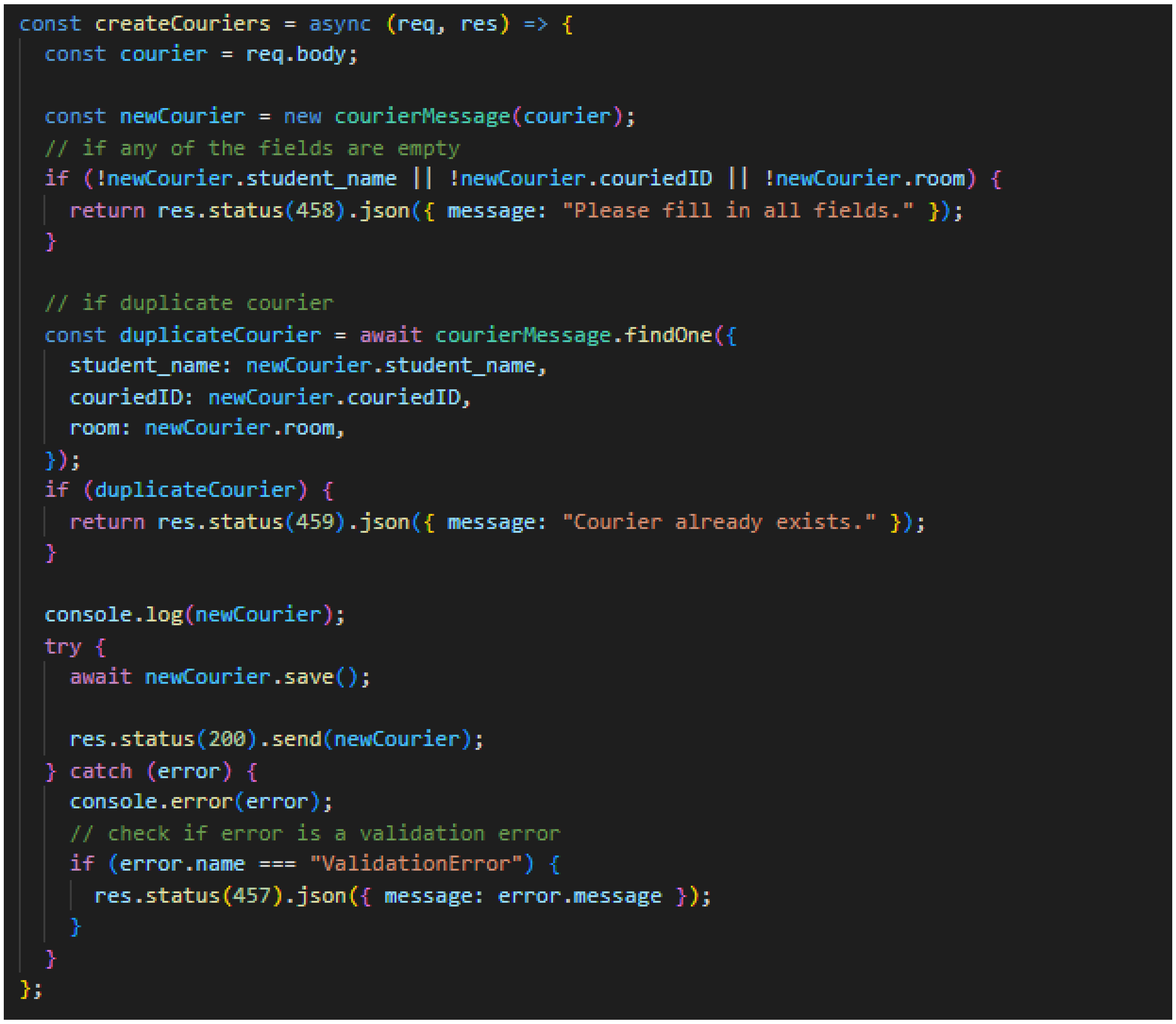
API To be Tested:



```
const getCouriers = async (req, res) => {
  try {
    const courierMessages = await courierMessage.find();
    // if not found
    if (courierMessages.length === 0) {
      return res.status(404).json({ message: "No couriers found." });
    }
    res.status(200).send(courierMessages);

  } catch (error) {
    res.status(409).send({ message: error.message });
  }
};
```

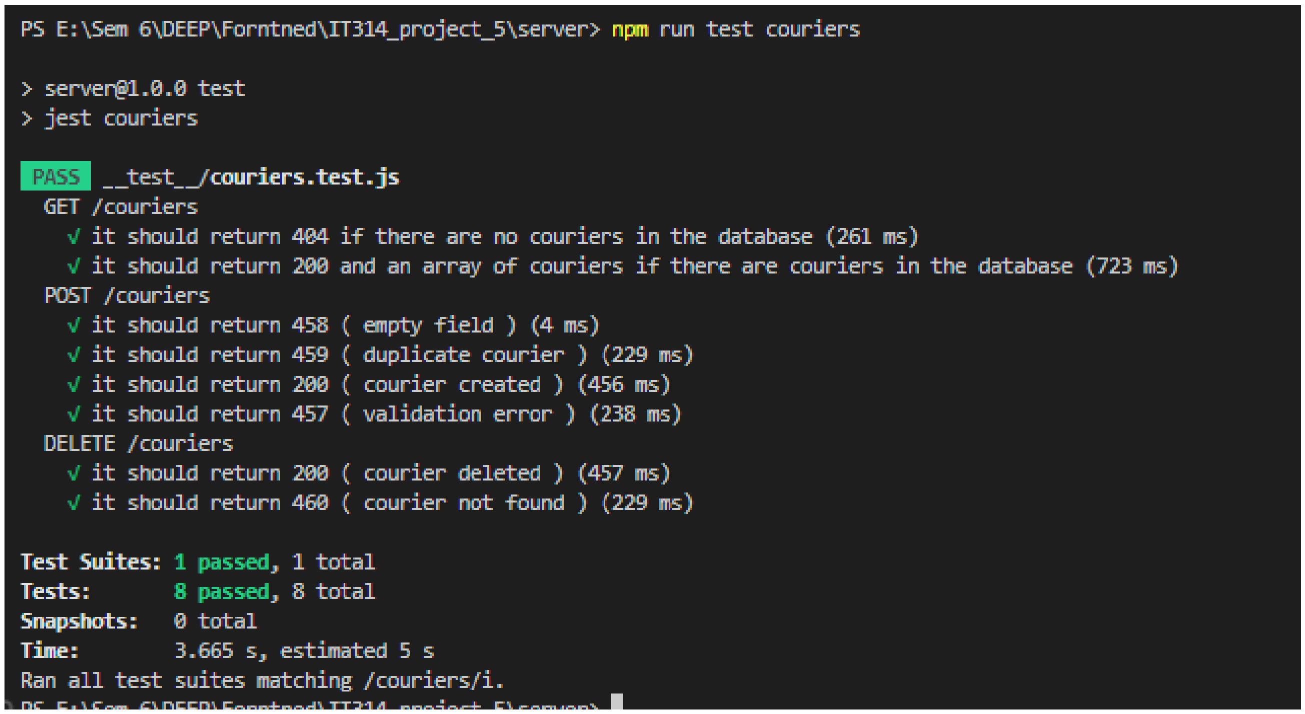


```
const deleteCouriers = async (req, res) => {

  // delete courier
  const courier = req.body;
  const courier_find = await courierMessage.findOne({
    student_name: courier.student_name,
    couriedID: courier.couriedID,
    room: courier.room,
  });
  if (!courier_find) {
    return res.status(460).json({ message: "Courier not found." });       You, 2 minutes ago • Uncommitte
  }
  try {
    await courierMessage.findByIdAndDelete(courier_find._id);
    res.status(200).json({ message: "Courier deleted successfully." });
  } catch (error) {
    res.status(409).json({ message: error.message });
  }
};
```



```
const createCouriers = async (req, res) => {
  const courier = req.body;

  const newCourier = new courierMessage(courier);
  // if any of the fields are empty
  if (!newCourier.student_name || !newCourier.couriedID || !newCourier.room) {
    return res.status(458).json({ message: "Please fill in all fields." });
  }

  // if duplicate courier
  const duplicateCourier = await courierMessage.findOne({
    student_name: newCourier.student_name,
    couriedID: newCourier.couriedID,
    room: newCourier.room,
  });
  if (duplicateCourier) {
    return res.status(459).json({ message: "Courier already exists." });
  }

  console.log(newCourier);
  try {
    await newCourier.save();

    res.status(200).send(newCourier);
  } catch (error) {
    console.error(error);
    // check if error is a validation error
    if (error.name === "ValidationError") {
      res.status(457).json({ message: error.message });
    }
  }
};
```

Test Results:



```
PS E:\Sem 6\DEEP\Forntned\IT314_project_5\server> npm run test couriers

> server@1.0.0 test
> jest couriers

 PASS  __test__/couriers.test.js
  GET /couriers
    √ it should return 404 if there are no couriers in the database (261 ms)
    √ it should return 200 and an array of couriers if there are couriers in the database (723 ms)
  POST /couriers
    √ it should return 458 ( empty field ) (4 ms)
    √ it should return 459 ( duplicate courier ) (229 ms)
    √ it should return 200 ( courier created ) (456 ms)
    √ it should return 457 ( validation error ) (238 ms)
  DELETE /couriers
    √ it should return 200 ( courier deleted ) (457 ms)
    √ it should return 460 ( courier not found ) (229 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        3.665 s, estimated 5 s
Ran all test suites matching /couriers/i.
PS E:\Sem 6\DEEP\Forntned\IT314_project_5\server>
```

Explanation :

- **Get request** :

1. Test case: No couriers found

This test case checks if the API returns a status code of 404 when No couriers found

2. Test case: courier successfully fetched

This test case checks if the API returns a status code of 200 when the courier can be fetched from DB successfully.

- **Post Request:**

1. Test case: If user post courier with empty field (title, message, creator)

This test case checks if the API returns a status code of 458 when user post courier with empty field.

2. Test case: Duplicate courier

This test case checks if the API returns a status code of 459 when user post a courier with fields that are already exist in the database.

3. Test Case: courier created Succesfully.

This test case checks if the API returns a status code of 200 when posted courier is stored at DB successfully.

4. Test Case: Validation Error

This test case checks if the API returns a status code of 457 when posted rules is not validated by Schema or DB. In this case, we post a courier with a title size< 3, which violates the schema condition that min length of a title is 3.

- **Delete Request:**

1. Test case: courier not found

This test case checks if the API returns a status code of 460 when the requested deleted courier doesn't exist in DB.

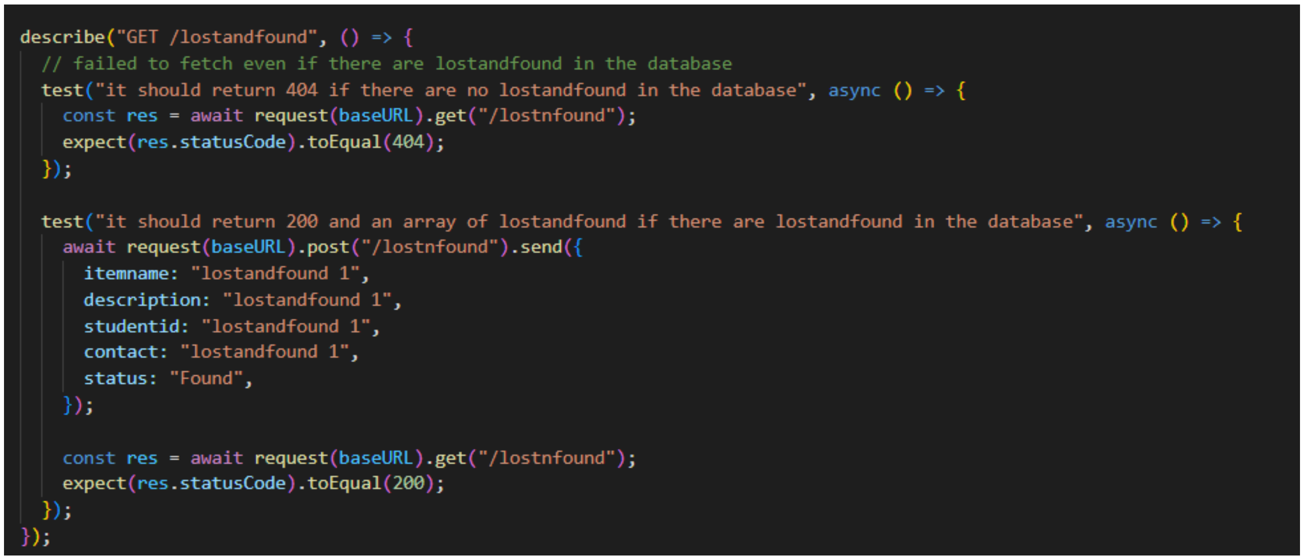
2. Test case: courier deleted successfully

This test case checks if the API returns a status code of 200 when the courier can be deleted from DB successfully.

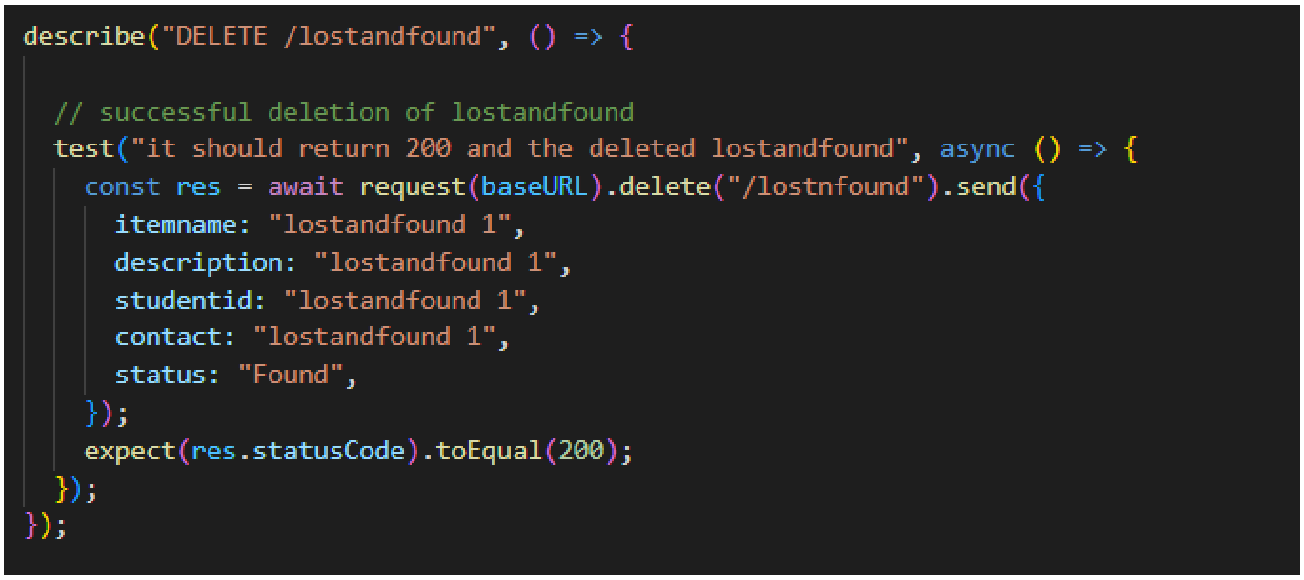
Time: 3.665 s, estimated 5 s

3. Lost and Found

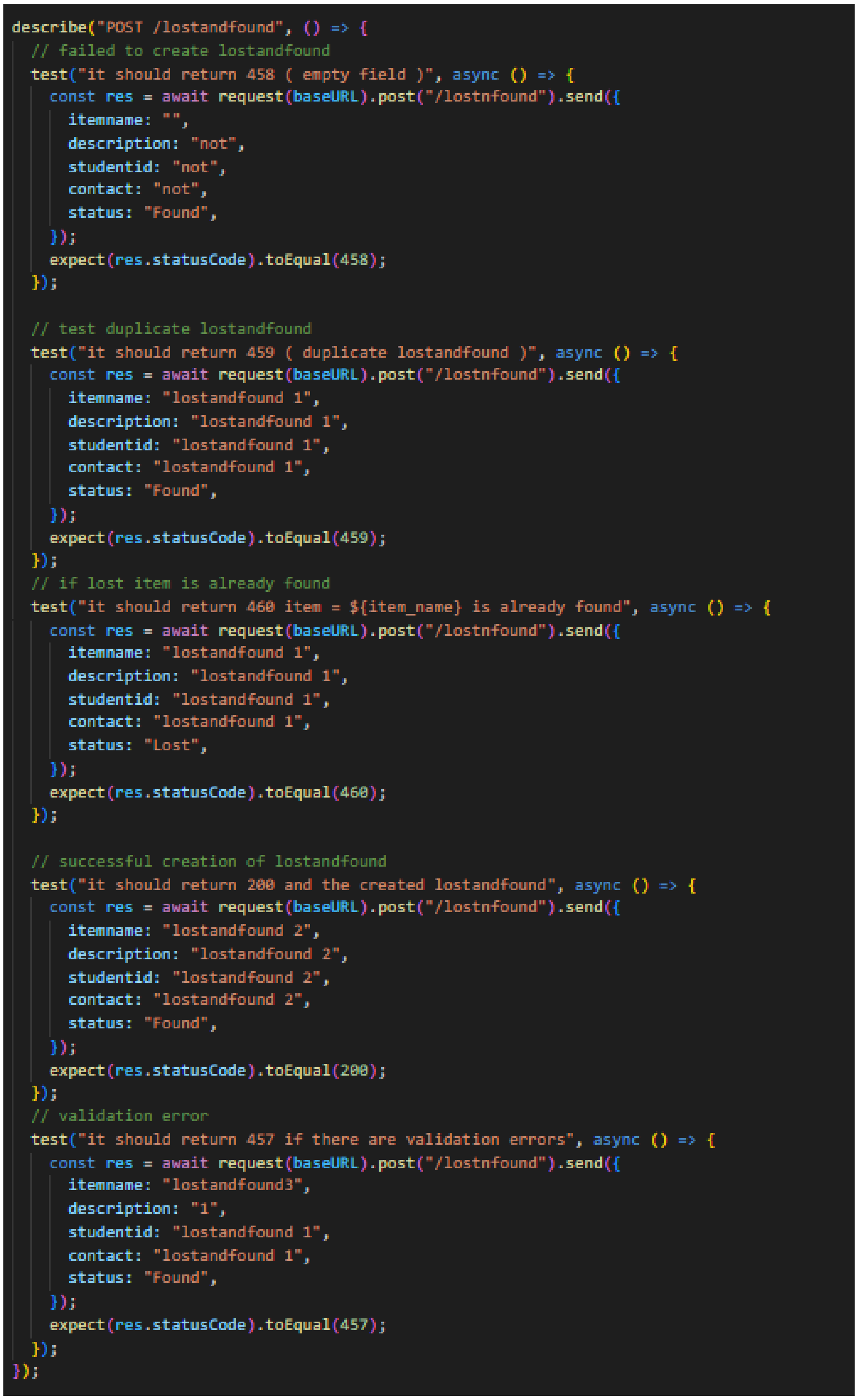
Test Codes:



```
describe("GET /lostandfound", () => {
  // failed to fetch even if there are lostandfound in the database
  test("it should return 404 if there are no lostandfound in the database", async () => {
    const res = await request(baseURL).get("/lostnfound");
    expect(res.statusCode).toEqual(404);
  });

  test("it should return 200 and an array of lostandfound if there are lostandfound in the database", async () => {
    await request(baseURL).post("/lostnfound").send({
      itemname: "lostandfound 1",
      description: "lostandfound 1",
      studentid: "lostandfound 1",
      contact: "lostandfound 1",
      status: "Found",
    });

    const res = await request(baseURL).get("/lostnfound");
    expect(res.statusCode).toEqual(200);
  });
});
```

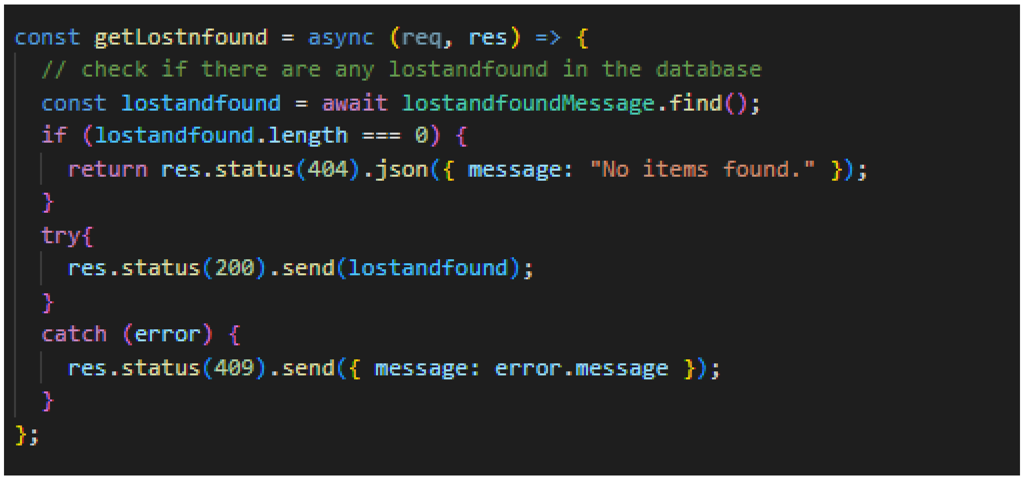


```
describe("DELETE /lostandfound", () => {

  // successful deletion of lostandfound
  test("it should return 200 and the deleted lostandfound", async () => {
    const res = await request(baseURL).delete("/lostnfound").send({
      itemname: "lostandfound 1",
      description: "lostandfound 1",
      studentid: "lostandfound 1",
      contact: "lostandfound 1",
      status: "Found",
    });
    expect(res.statusCode).toEqual(200);
  });
});
```

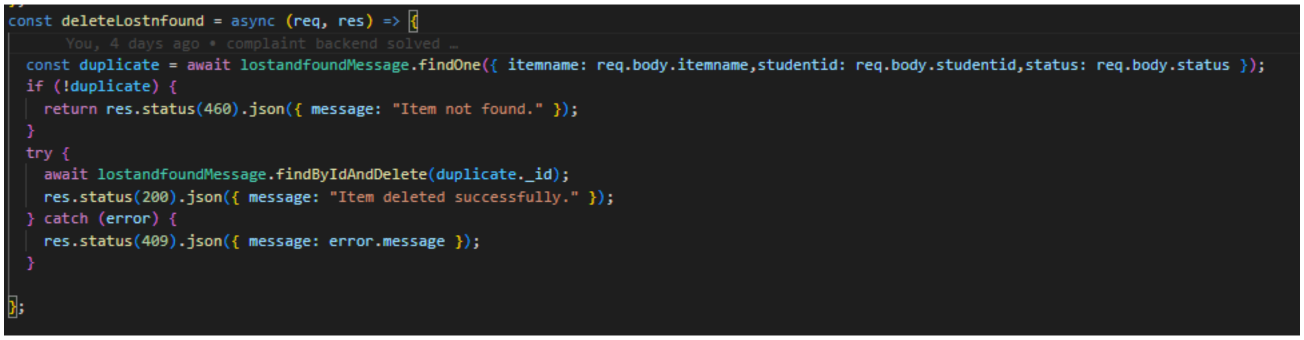


```
describe("POST /lostandfound", () => {
  // failed to create lostandfound
  test("it should return 458 ( empty field )", async () => {
    const res = await request(baseURL).post("/lostnfound").send({
      itemname: "",
      description: "not",
      studentid: "not",
      contact: "not",
      status: "Found",
    });
    expect(res.statusCode).toEqual(458);
  });

  // test duplicate lostandfound
  test("it should return 459 ( duplicate lostandfound )", async () => {
    const res = await request(baseURL).post("/lostnfound").send({
      itemname: "lostandfound 1",
      description: "lostandfound 1",
      studentid: "lostandfound 1",
      contact: "lostandfound 1",
      status: "Found",
    });
    expect(res.statusCode).toEqual(459);
  });
  // if lost item is already found
  test("it should return 460 item = ${item_name} is already found", async () => {
    const res = await request(baseURL).post("/lostnfound").send({
      itemname: "lostandfound 1",
      description: "lostandfound 1",
      studentid: "lostandfound 1",
      contact: "lostandfound 1",
      status: "Lost",
    });
    expect(res.statusCode).toEqual(460);
  });

  // successful creation of lostandfound
  test("it should return 200 and the created lostandfound", async () => {
    const res = await request(baseURL).post("/lostnfound").send({
      itemname: "lostandfound 2",
      description: "lostandfound 2",
      studentid: "lostandfound 2",
      contact: "lostandfound 2",
      status: "Found",
    });
    expect(res.statusCode).toEqual(200);
  });
  // validation error
  test("it should return 457 if there are validation errors", async () => {
    const res = await request(baseURL).post("/lostnfound").send({
      itemname: "lostandfound3",
      description: "1",
      studentid: "lostandfound 1",
      contact: "lostandfound 1",
      status: "Found",
    });
    expect(res.statusCode).toEqual(457);
  });
});
```

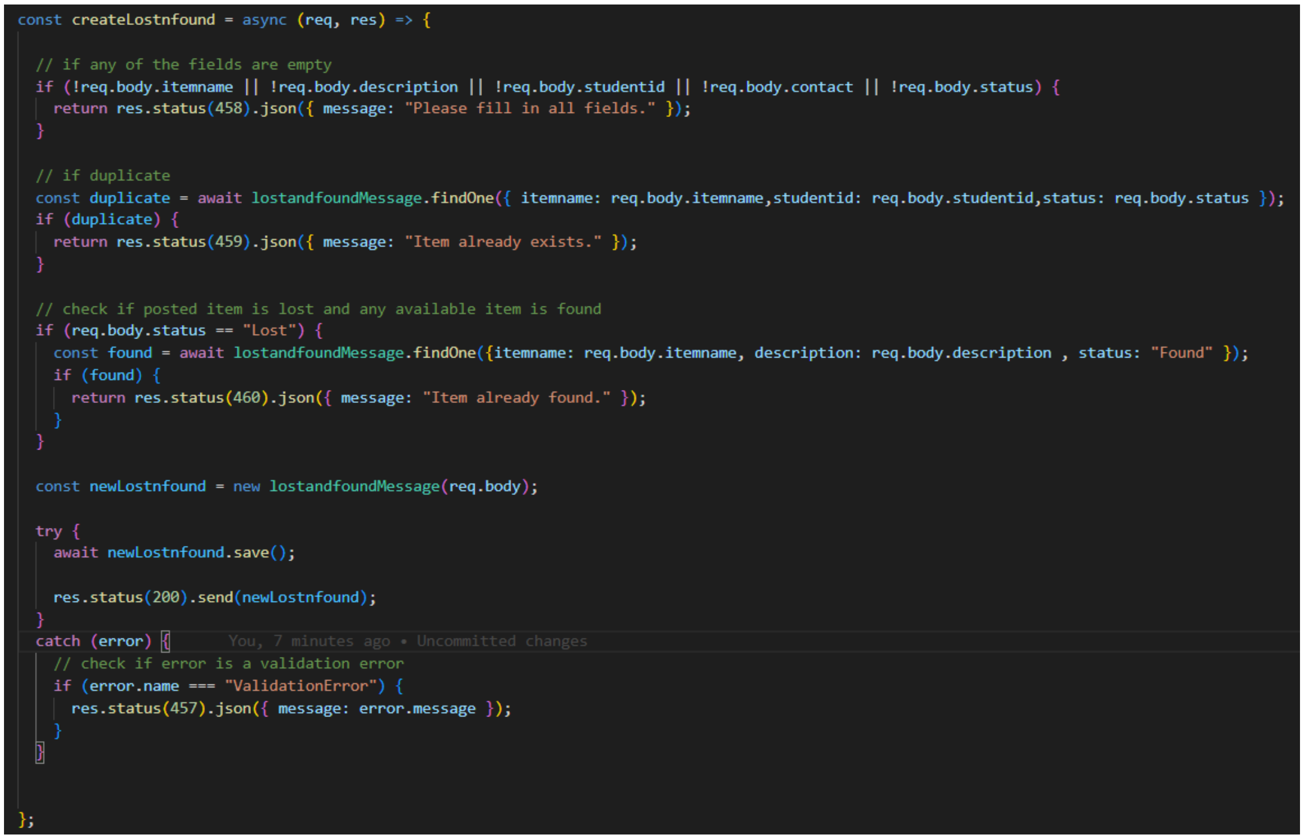
<u>API To be Tested:</u>



```
const getLostnfound = async (req, res) => {
  // check if there are any lostandfound in the database
  const lostandfound = await lostandfoundMessage.find();
  if (lostandfound.length === 0) {
    return res.status(404).json({ message: "No items found." });
  }
  try{
    res.status(200).send(lostandfound);
  }
  catch (error) {
    res.status(409).send({ message: error.message });
  }
};
```

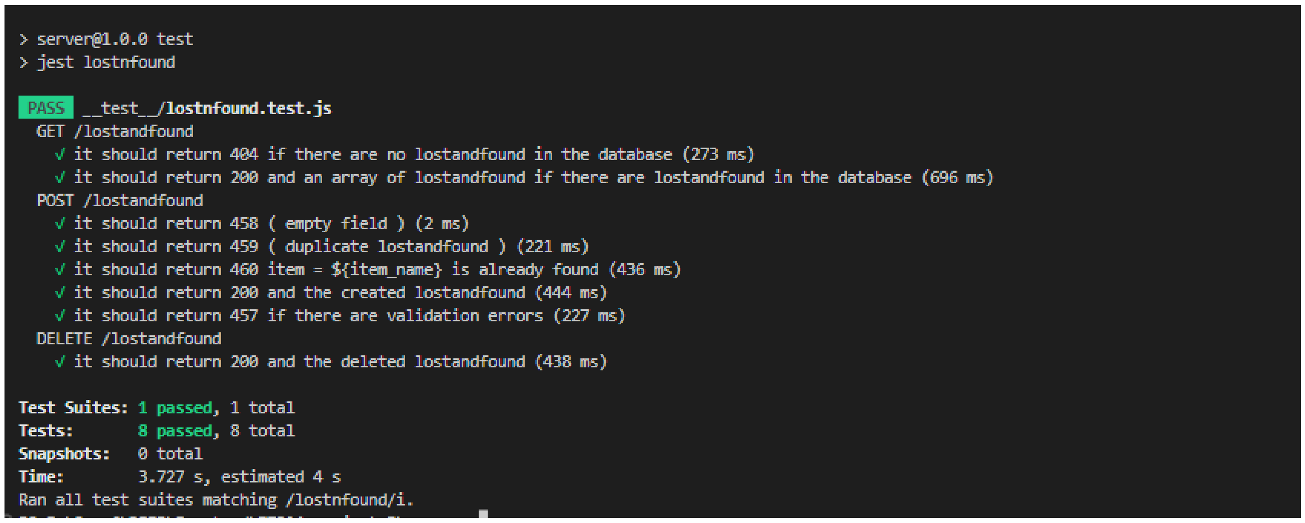


```
const deleteLostnfound = async (req, res) => {
      You, 4 days ago • complaint backend solved …
  const duplicate = await lostandfoundMessage.findOne({ itemname: req.body.itemname,studentid: req.body.studentid,status: req.body.status });
  if (!duplicate) {
    return res.status(460).json({ message: "Item not found." });
  }
  try {
    await lostandfoundMessage.findByIdAndDelete(duplicate._id);
    res.status(200).json({ message: "Item deleted successfully." });
  } catch (error) {
    res.status(409).json({ message: error.message });
  }

};
```



```
const createLostnfound = async (req, res) => {

  // if any of the fields are empty
  if (!req.body.itemname || !req.body.description || !req.body.studentid || !req.body.contact || !req.body.status) {
    return res.status(458).json({ message: "Please fill in all fields." });
  }

  // if duplicate
  const duplicate = await lostandfoundMessage.findOne({ itemname: req.body.itemname,studentid: req.body.studentid,status: req.body.status });
  if (duplicate) {
    return res.status(459).json({ message: "Item already exists." });
  }

  // check if posted item is lost and any available item is found
  if (req.body.status == "Lost") {
    const found = await lostandfoundMessage.findOne({itemname: req.body.itemname, description: req.body.description , status: "Found" });
    if (found) {
      return res.status(460).json({ message: "Item already found." });
    }
  }

  const newLostnfound = new lostandfoundMessage(req.body);

  try {
    await newLostnfound.save();

    res.status(200).send(newLostnfound);
  }
  catch (error) {        You, 7 minutes ago • Uncommitted changes
    // check if error is a validation error
    if (error.name === "ValidationError") {
      res.status(457).json({ message: error.message });
    }
  }


};
```

Test Results:



```
> server@1.0.0 test
> jest lostnfound

 PASS  __test__/lostnfound.test.js
  GET /lostandfound
    √ it should return 404 if there are no lostandfound in the database (273 ms)
    √ it should return 200 and an array of lostandfound if there are lostandfound in the database (696 ms)
  POST /lostandfound
    √ it should return 458 ( empty field ) (2 ms)
    √ it should return 459 ( duplicate lostandfound ) (221 ms)
    √ it should return 460 item = ${item_name} is already found (436 ms)
    √ it should return 200 and the created lostandfound (444 ms)
    √ it should return 457 if there are validation errors (227 ms)
  DELETE /lostandfound
    √ it should return 200 and the deleted lostandfound (438 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        3.727 s, estimated 4 s
Ran all test suites matching /lostnfound/i.
```

Explanation :

- **Get request** :

1. Test case: No lostnfound found

This test case checks if the API returns a status code of 404 when No lostnfound found

2. Test case: Complaint successfully fetched

This test case checks if the API returns a status code of 200 when the lostnfound can be fetched from DB successfully.

- **Post Request:**

1. Test case: Empty Field

This test case checks if the API returns a status code of 458 when user post Complaint with empty field (title, message, creator)

2. Test case: Duplicate Complaint

This test case checks if the API returns a status code of 459 when user post a complaint with fields that are already exist in the database.

3. Test Case: item is already found

This test case checks if the API returns a status code of 460 when user post item as lost and that item already exists as found.

4. Test Case: Complaint created Succesfully.

This test case checks if the API returns a status code of 200 when posted lostnfound is stored at DB successfully.

5. Test Case: Validation Error

This test case checks if the API returns a status code of 457 when posted rules is not validated by Schema or DB. In this case, we post a lostnfound with a title size< 3, which violates the schema condition that min length of a itemname is 3.

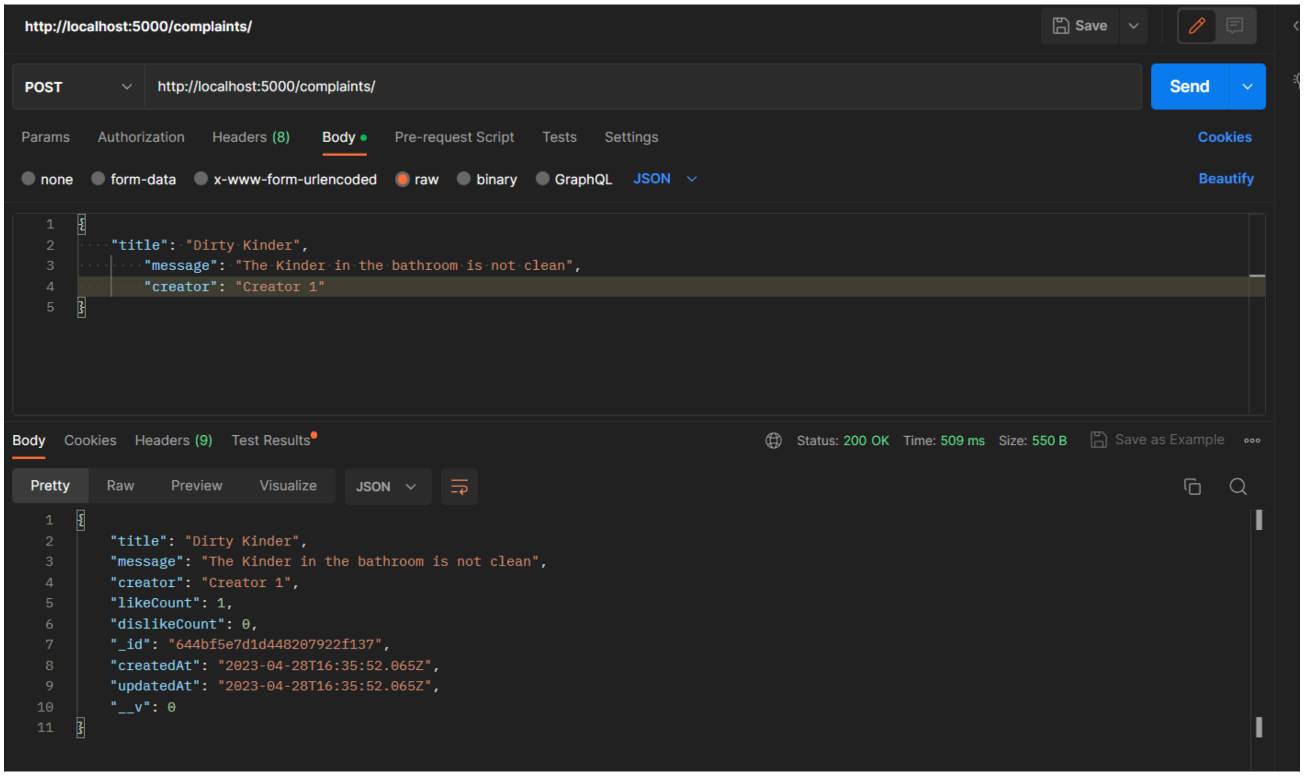
- **Delete Request:**

1. Test case: lostnfound deleted successfully

This test case checks if the API returns a status code of 200 when the complaint can be deleted from DB successfully.

Time: 3.727 s, estimated 4 s

- **Postman Screenshots during Development:**

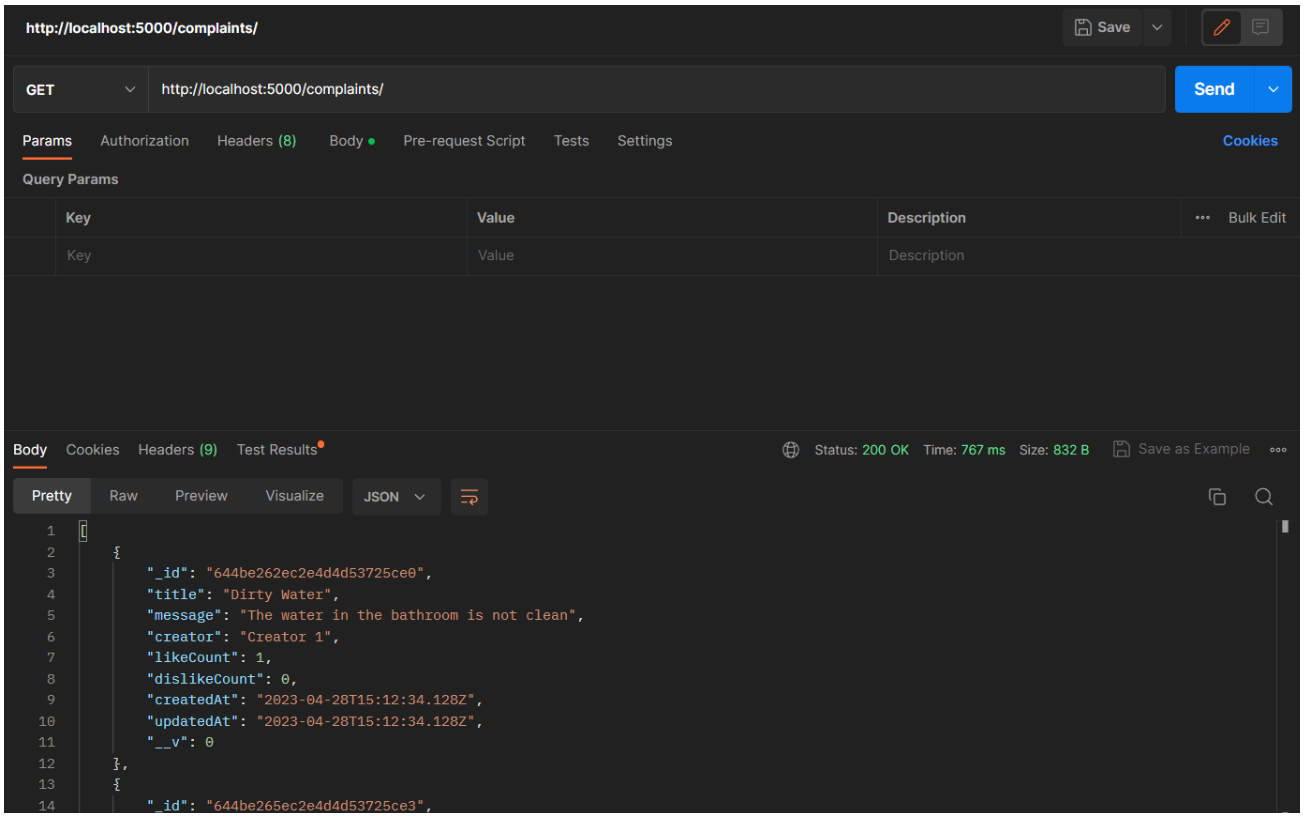


POST http://localhost:5000/complaints/

```
{
    "title": "Dirty Kinder",
        "message": "The Kinder in the bathroom is not clean",
        "creator": "Creator 1"
}
```

Status: 200 OK Time: 509 ms Size: 550 B

```
{
    "title": "Dirty Kinder",
    "message": "The Kinder in the bathroom is not clean",
    "creator": "Creator 1",
    "likeCount": 1,
    "dislikeCount": 0,
    "_id": "644bf5e7d1d448207922f137",
    "createdAt": "2023-04-28T16:35:52.065Z",
    "updatedAt": "2023-04-28T16:35:52.065Z",
    "__v": 0
}
```



GET http://localhost:5000/complaints/

Key	Value	Description
Key	Value	Description

Status: 200 OK Time: 767 ms Size: 832 B

```
[
    {
        "_id": "644be262ec2e4d4d53725ce0",
        "title": "Dirty Water",
        "message": "The water in the bathroom is not clean",
        "creator": "Creator 1",
        "likeCount": 1,
        "dislikeCount": 0,
        "createdAt": "2023-04-28T15:12:34.128Z",
        "updatedAt": "2023-04-28T15:12:34.128Z",
        "__v": 0
    },
    {
        "_id": "644be265ec2e4d4d53725ce3",
```