Unit Testing Report:

The techstack used in backend is Express(framework of Node), HTML,CSS,Bootstrap and pure javascript in frontend and Mongodb in database.

The backend systems were tested using mocha and chai modules. These libraries provide a way to test out RESTful APIs and help pointing out mistakes that can potentially lead to break code further. Through this testing, we found many test cases that were causing some internal issues and we figured out a way to deal with those edge cases, which helped our code become more robust and better in terms of error handling.

Let’s take a look at a few examples of the testing we did using these libraries:

Let's try to test the RESTful API of courses in our web application:
Here are some test cases:

```
//first one refers the new object to be created and the second one referes the updated one
const testCases = [
  [
    {
      courseName: "testCourse1",
      courseDescription: "test content1",
    },
    {
      courseName: "testCourse1.1",
      courseDescription: "test content 1.1",
    },
  ],
  [
    {
      courseName: "testCourse2",
      courseDescription: "test content2",
    },
    {
      courseName: "testCourse1.1",
      courseDescription: "test content 2.1",
    },
  ],
];

module.exports = { testCases };
```

This list is certainly not exhaustive, we can come up with a lot of test cases. It's just one of those test cases we used to test our API.

Now almost all of our routes require user login. So we started with that as shown so that we can add some statefulness in consequent HTTP requests by sending the cookie:

```
describe("Testing Course", () => {
  for (let index = 0; index < testCases.length; index++) {
    let cookie;
    //login module
    describe("POST /login", () => {
      it("it should login user", (done) => {
        console.log(`\nRunning testcase: #${index + 1}\n`);
        chai
          .request(app)
          .post("/login")
          .send({
            username_email: "kavan",
            password: "kavan",
          })
          .end((err, res) => {
            res.statusCode.should.equal(200);
            res.body.should.be.a("object");
            cookie = res.headers["set-cookie"][0];
            done();
          });
      });
    });
```

Test for GET /courses:

```
//read course module
describe("GET /courses", () => {
  it("it should read courses", (done) => {
    chai
      .request(app)
      .get("/courses")
      .set("Cookie", cookie)
      .end((err, res) => {
        res.statusCode.should.equal(200);
        res.body.should.be.a("object");
        done();
      });
  });
});
```

Test for POST /courses:

```
//add course module
describe("POST /courses", () => {
  it("it should add courses", (done) => {
    chai
      .request(app)
      .post("/courses")
      .set("Cookie", cookie)
      .send(testCases[index][0])
      .end((err, res) => {
        res.statusCode.should.equal(200);
        res.body.should.be.a("object");
        done();
      });
  });
});
```

Test for PATCH /courses/:id

```
//update course
describe("PATCH /courses/:id", () => {
  it("it should update course", (done) => {
    Course.findOne(testCases[index][0]).then((res) => {
      chai
        .request(app)
        .patch(`/courses/${res._id.toString()}`)
        .set("Cookie", cookie)
        .send(testCases[index][1])
        .end((err, res) => {
          res.statusCode.should.equal(200);
          res.body.should.be.a("object");
          done();
        });
    });
  });
});
```

Test for DELETE /courses/:id

```
    //delete course
    describe("DELETE /courses/:id", () => {
      it("it should delete course", (done) => {
        Course.findOne(testCases[index][1]).then((res) => {
          chai
            .request(app)
            .delete(`/courses/${res._id.toString()}`)
            .set("Cookie", cookie)
            .end((err, res) => {
              res.statusCode.should.equal(200);
              res.body.should.be.a("object");
              done();
            });
        });
      });
    });
  }
});
```

We are not using beforeEach or afterEach to perform operations before we start/finish a test because we are creating one instance, updating it and deleting it in the same test. So technically we are not manipulating the database as long as each test case evaluates to be passed.

After running the test file….

```
> 2023@1.0.0 test
> mocha --timeout 1000000 test/course/course.test.js



  Testing Course
    POST /login

Running testcase: #1

Connect to DB Succesfully!!
      ✔ it should login user (8408ms)
    GET /courses
      ✔ it should read courses (1128ms)
    POST /courses
      ✔ it should add courses (1434ms)
    PATCH /courses/:id
      ✔ it should update course (1308ms)
    DELETE /courses/:id
      ✔ it should delete course (1664ms)
    POST /login

Running testcase: #2

      ✔ it should login user (1316ms)
    GET /courses
      ✔ it should read courses (906ms)
    POST /courses
      ✔ it should add courses (823ms)
    PATCH /courses/:id
      ✔ it should update course (1225ms)
    DELETE /courses/:id
      ✔ it should delete course (1495ms)


  10 passing (20s)
```

We have done this test for a few other collections as well. The test files are available in Github Repository.
Here are the results we got from courseContent API:

```
> 2023@1.0.0 test
> mocha --timeout 1000000 test/courseContent/content.test.js



  Testing Course Content
    POST /login

Running testcase: #1

Connect to DB Succesfully!!
      ✔ it should login user (6001ms)
    GET /courses/view
      ✔ it should read contents of a course (1217ms)
    POST /courses/add/:id
      ✔ it should add content (1529ms)
    PATCH /courses/update/:id
      ✔ it should update content (1200ms)
    DELETE /courses/delete/:id
      ✔ it should delete content (950ms)
    POST /login

Running testcase: #2

      ✔ it should login user (1036ms)
    GET /courses/view
      ✔ it should read contents of a course (905ms)
    POST /courses/add/:id
      ✔ it should add content (1124ms)
    PATCH /courses/update/:id
      ✔ it should update content (1156ms)
    DELETE /courses/delete/:id
      ✔ it should delete content (894ms)


  10 passing (16s)
```

Here are the results from register/login the user: