IDG2022 REPORT

Marie Holme Kjær

mariehkj@stud.ntnu.no

Studentnr: 539668

Truls Teige Pettersen

trulstp@stud.ntnu.no

Studentnr: 539673

We chose to split up our front end and back end parts of the project due to issues with having the PORT of the server running at the same time as the PORT for the front end react application.

Front End

https://github.com/trulstp/Cloud

https://stark-island-14644.herokuapp.com/

Back End

https://github.com/trulstp/registrationbackend

https://afternoon-peak-35116.herokuapp.com/

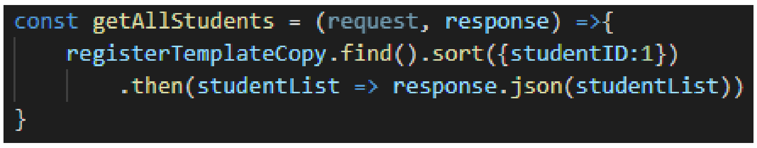
Work in progress git that we ended up abandoning:

https://github.com/trulstp/Webprosjekt

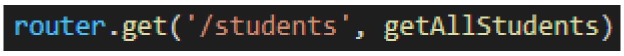
System explanation

Our system can view all students or just view one, delete a student or update the data about a student. You also have the ability to create a new student. To keep everything structured we have one file for the routes and one file for the controllers.

View all students



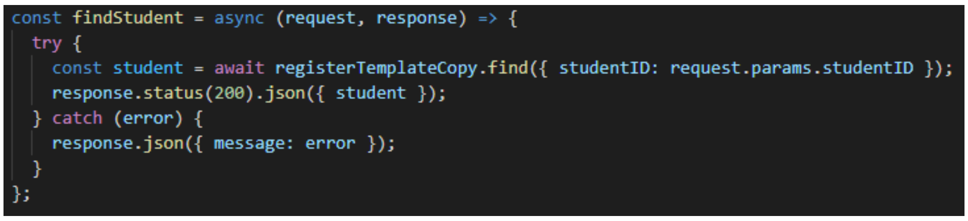
```
const getAllStudents = (request, response) =>{
    registerTemplateCopy.find().sort({studentID:1})
        .then(studentList => response.json(studentList))
}
```



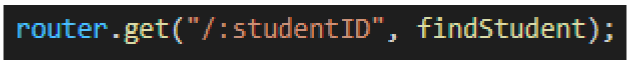
```
router.get('/students', getAllStudents)
```

Here is our code to view all students. We start by requesting the schema, which contains ID, name etc. By using “find” we request this data from the database. Then we use “sort” to sort the students by their ID in ascending order. We then put the response in “/students” and gather the data in the front-end as a JSON format, which in turn puts the data in a table.

Search for student



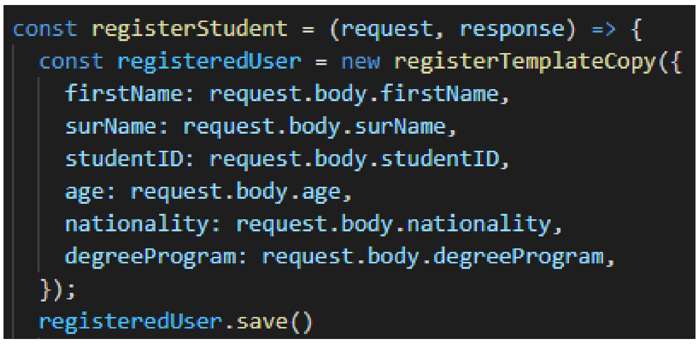
```
const findStudent = async (request, response) => {
  try {
    const student = await registerTemplateCopy.find({ studentID: request.params.studentID });
    response.status(200).json({ student });
  } catch (error) {
    response.json({ message: error });
  }
};
```



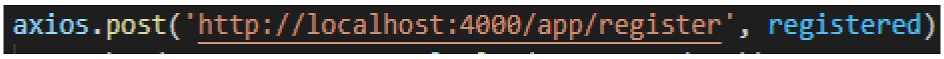
```
router.get("/:studentID", findStudent);
```

To find one specific student, we search the database for a student with the same studentID as we requested in an input field. To do this we use “find” with the studentID as a parameter. To get this parameter, we use axios to store the studentID in the URL.

Create student



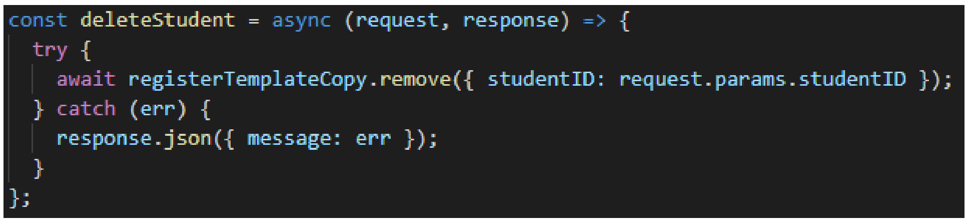
```
const registerStudent = (request, response) => {
  const registeredUser = new registerTemplateCopy({
    firstName: request.body.firstName,
    surName: request.body.surName,
    studentID: request.body.studentID,
    age: request.body.age,
    nationality: request.body.nationality,
    degreeProgram: request.body.degreeProgram,
  });
  registeredUser.save()
```



```
axios.post('http://localhost:4000/app/register', registered)
```

To create a new student entry axios sends data from the form to “/register”. From there the back-end creates a new instance of the “registerTemplateCopy” schema, where it stores said data in the correct format. We then use “save” to store the new student entry in MongoDB.

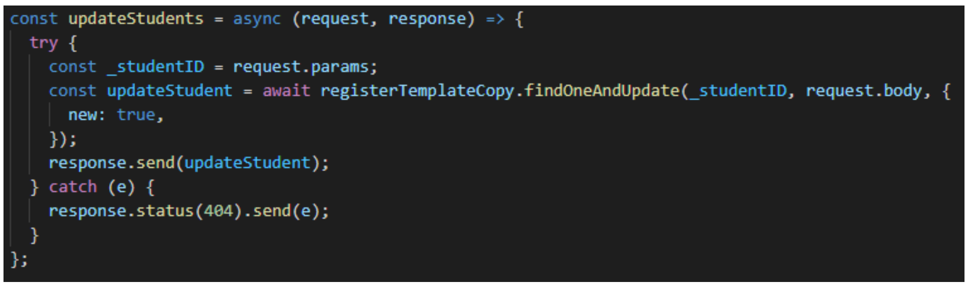
Delete student



```
const deleteStudent = async (request, response) => {
  try {
    await registerTemplateCopy.remove({ studentID: request.params.studentID });
  } catch (err) {
    response.json({ message: err });
  }
};
```

Delete student is very much alike Find student. We use “remove” with studentID as the parameter to delete that specific entry from the database.

Update student



```
const updateStudents = async (request, response) => {
  try {
    const _studentID = request.params;
    const updateStudent = await registerTemplateCopy.findOneAndUpdate(_studentID, request.body, {
      new: true,
    });
    response.send(updateStudent);
  } catch (e) {
    response.status(404).send(e);
  }
};
```

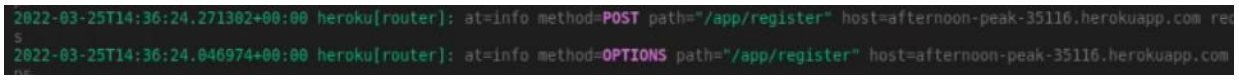
Update student finds the requested student by the same means as Delete- and Find student. We then use “findOneAndUpdate” with studentID and the new data written in the form as parameters. We then send the updated entry to the database.

Latency Measurement

Here are our latency measurements.

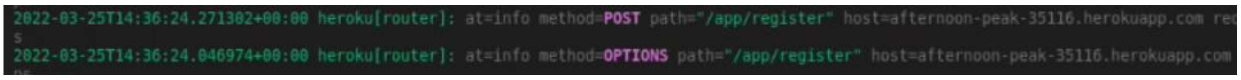
Desktop

Backend:



```
2022-03-25T14:36:24.271302+00:00 heroku[router]: at=info method=POST path="/app/register" host=afternoon-peak-35116.herokuapp.com req
s
2022-03-25T14:36:24.046974+00:00 heroku[router]: at=info method=OPTIONS path="/app/register" host=afternoon-peak-35116.herokuapp.com
```

Frontend:



```
2022-03-25T14:36:24.271302+00:00 heroku[router]: at=info method=POST path="/app/register" host=afternoon-peak-35116.herokuapp.com req
s
2022-03-25T14:36:24.046974+00:00 heroku[router]: at=info method=OPTIONS path="/app/register" host=afternoon-peak-35116.herokuapp.com
```

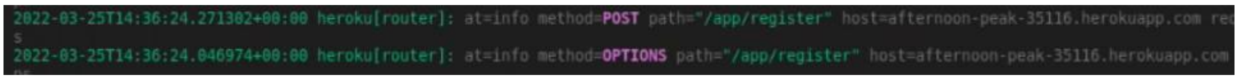
Round-trip delay:

(t_3 – t_0) - (t_2 – t_1)

(227.812 - 109.747) - (54 810.516 - 54 559.459) = -132.992 milliseconds

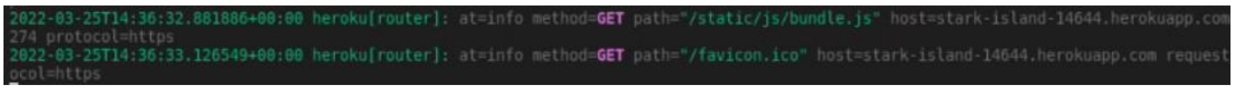
Mobile on WIFI

Backend:



```
2022-03-25T14:36:24.271302+00:00 heroku[router]: at=info method=POST path="/app/register" host=afternoon-peak-35116.herokuapp.com req
s
2022-03-25T14:36:24.046974+00:00 heroku[router]: at=info method=OPTIONS path="/app/register" host=afternoon-peak-35116.herokuapp.com
```

Frontend:



```
2022-03-25T14:36:32.881886+00:00 heroku[router]: at=info method=GET path="/static/js/bundle.js" host=stark-island-14644.herokuapp.com
274 protocol=https
2022-03-25T14:36:33.126549+00:00 heroku[router]: at=info method=GET path="/favicon.ico" host=stark-island-14644.herokuapp.com request
ocol=https
```

Round-trip delay:

(t_3 – t_0) - (t_2 – t_1)

(33 126.549 - 32 881.886) - (24 271.302 - 24 046.974) = 20.335 milliseconds