

**FACULTY OF ENGINEERING**

**REI 414  
Practical:  
E-Learning Platform**

by:

**Jacques Beukes    26028107**

**Dewald Krynauw    26013835**

Submitted in pursuit of the degree

**BACHELOR OF ENGINEERING**

**In**

**COMPUTER AND ELECTRONIC ENGINEERING**

**North-West-University Potchefstroom Campus**

Supervisor: Mr. A. Alberts  
Potchefstroom  
2018

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>1</b>
2.1	MySQL . . . . .	1
2.2	Node.js . . . . .	1
2.3	HTML . . . . .	1
2.4	CSS . . . . .	1
2.5	Javascript . . . . .	1
<b>3</b>	<b>Database Structure</b>	<b>2</b>
<b>4</b>	<b>Front-end</b>	<b>3</b>
4.1	Login and Register Interface . . . . .	3
4.2	Lecturer Interface . . . . .	4
4.3	Student Interface . . . . .	6
<b>5</b>	<b>Back-end</b>	<b>7</b>
5.1	Sign up and Login . . . . .	7
5.2	Courses . . . . .	8
5.3	Lessons . . . . .	9
5.4	Assessments . . . . .	10
5.5	Grade-book . . . . .	11
<b>6</b>	<b>Conclusion</b>	<b>12</b>
6.1	Strenghts . . . . .	12
6.2	Flaws . . . . .	12
6.3	Improvements . . . . .	12
6.4	Techniques Learned . . . . .	13

## References

13

## Abbreviations

ERD	Entity Relationship Diagram
DBMS	Database Management System
SQL	Structured Query Language

## List of Figures

3.1 Database ERD . . . . .	2
4.1 Login page . . . . .	3
4.2 Register page . . . . .	3
4.3 Lecturer profile page . . . . .	4
4.4 Page for creating a course . . . . .	4
4.5 Page for adding a course . . . . .	4
4.6 Lecturer lessons page . . . . .	5
4.7 Lecturer assessments page . . . . .	5
4.8 Lecturer gradebook page . . . . .	5
4.9 Creating assessments page . . . . .	6
4.10 Update student marks page . . . . .	6
4.11 Student profile page . . . . .	6
5.1 Code: Local Sign-up . . . . .	7
5.2 Code: Local Login . . . . .	8
5.3 Code: Create new course . . . . .	8
5.4 Code: Get Courses . . . . .	9
5.5 Code: Create Lessons . . . . .	10
5.6 Code: Create and Update Marks for Assessments . . . . .	11
5.7 Code: Grade book . . . . .	12

## List of Tables

# 1 Introduction

In this practical, an e-learning website is created to serve as a platform for communicating course material and evaluation results between lecturers and students. This document is the report for the practical. It contains a description of the website's database, front-end and back-end structure.

## 2 Background

### 2.1 MySQL

MySQL is a common open-source relational database management system (DBMS). Like most DBMS, MySQL follows a client-server approach to maintain the database. A client connects to a server and interacts with the database by sending SQL commands to the server. A corresponding MySQL Node package was used by the webserver to interface with the database.

### 2.2 Node.js

Node.js is an open-source JavaScript runtime built on Chrome's V8 JavaScript engine. In other words, it is a cross-platform environment that executes JavaScript code out of the browser. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

The webserver of this practical is implemented in Node. Libraries are dubbed as packages in the Node environment. Node's package management system (NPM) are used for the management of server packages.

### 2.3 HTML

### 2.4 CSS

### 2.5 Javascript

### 3 Database Structure

An Entity Relationship Diagram (ERD) of the website's database structure is provided in figure 3.1.

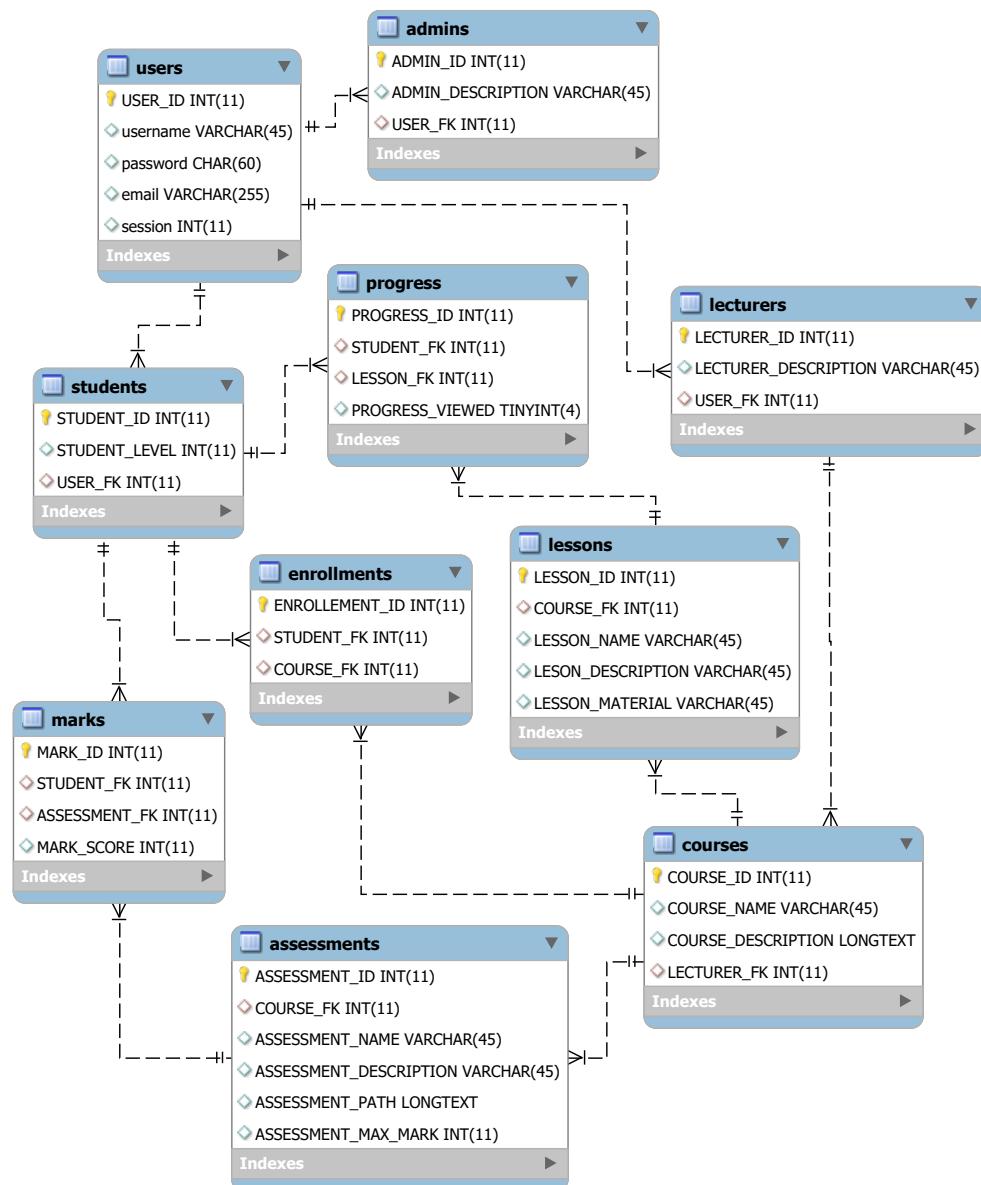


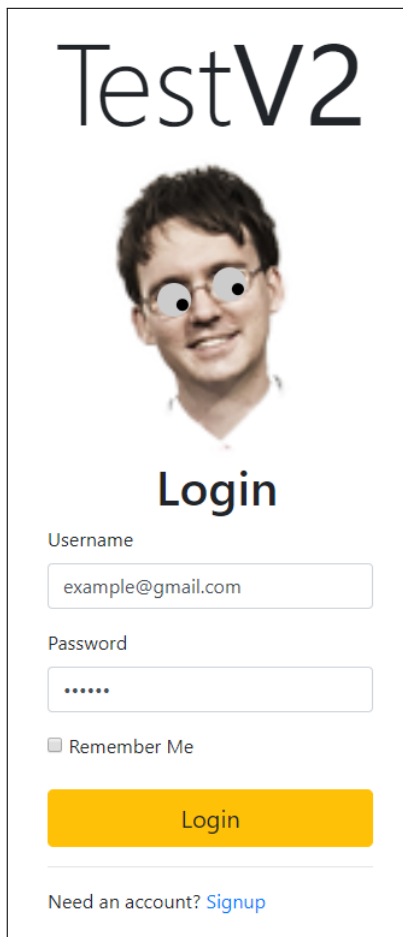
Figure 3.1: Database ERD

## 4 Front-end

The following section provide images of the website's user interface to illustrate the front-end of the completed site.

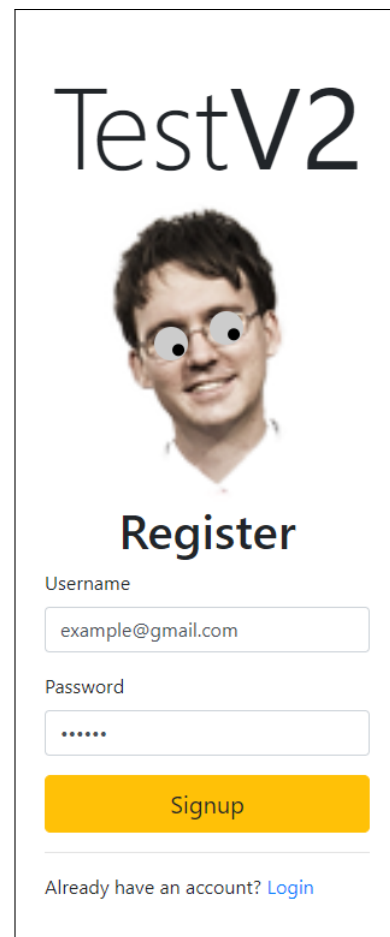
### 4.1 Login and Register Interface

The first page any user encounters is the login page (figure 4.1). If the current user does not have an account, the *Signup*-link can be used to browse to the registration page (figure 4.2).



The login page features a large heading 'TestV2' at the top. Below it is a cartoon illustration of a man with glasses and a white shirt. Under the illustration is the word 'Login'. The form includes a 'Username' field with the text 'example@gmail.com', a 'Password' field with six dots, a 'Remember Me' checkbox, and a yellow 'Login' button. At the bottom, there is a link that says 'Need an account? Signup'.

Figure 4.1: Login page



The register page features a large heading 'TestV2' at the top. Below it is the same cartoon illustration of a man with glasses and a white shirt. Under the illustration is the word 'Register'. The form includes a 'Username' field with the text 'example@gmail.com', a 'Password' field with six dots, and a yellow 'Signup' button. At the bottom, there is a link that says 'Already have an account? Login'.

Figure 4.2: Register page

## 4.2 Lecturer Interface

Upon signing in, lecturers are routed to the profile page where they can choose to create or add courses (figures 4.4 and 4.5). If the lecturer clicks on one of the added courses, he is taken to a corresponding section for managing the particular course.

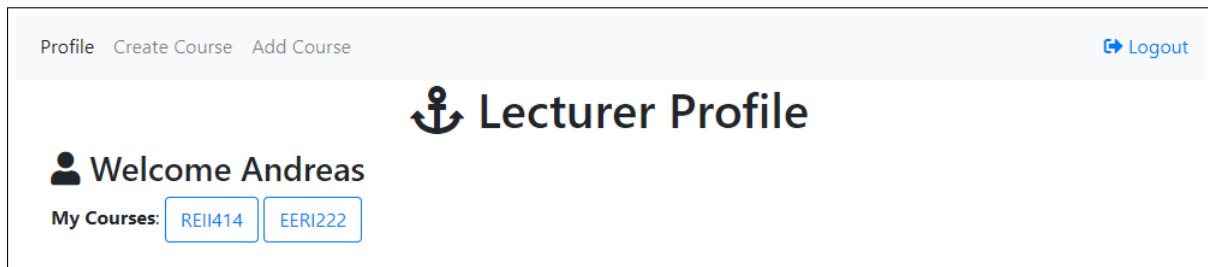


Figure 4.3: Lecturer profile page

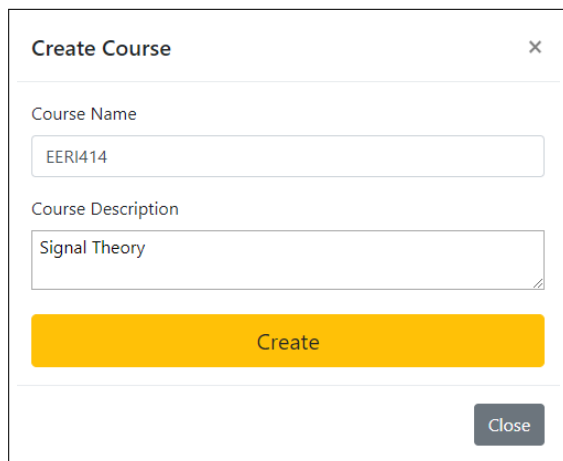


Figure 4.4: Page for creating a course

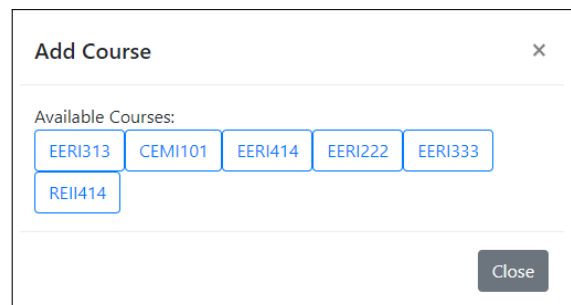


Figure 4.5: Page for adding a course


Figure 4.6 shows the page for creating and deleting lessons. The page shown in figure 4.7 is used to create new assessments (figure 4.9) and update student marks (figure 4.10). The page in figure 4.8 is used to view each student's marks for every assessment.



Profile Lessons Assessments Gradebook [Logout](#)

## REII414

*Database & Philosophy*

#	Lesson	Description	Material	
2	 Tutorial	ERD	Tut problems 2018-02-15.txt	<a href="#">Delete</a>

[Create Lesson](#)

Figure 4.6: Lecturer lessons page

Profile Lessons Assessments Gradebook [Logout](#)

## REII414

Name	Description	Maximum Marks	
Test 1	Super Easy	20	<a href="#">Update Marks</a>

[Create Assessment](#)

Figure 4.7: Lecturer assessments page

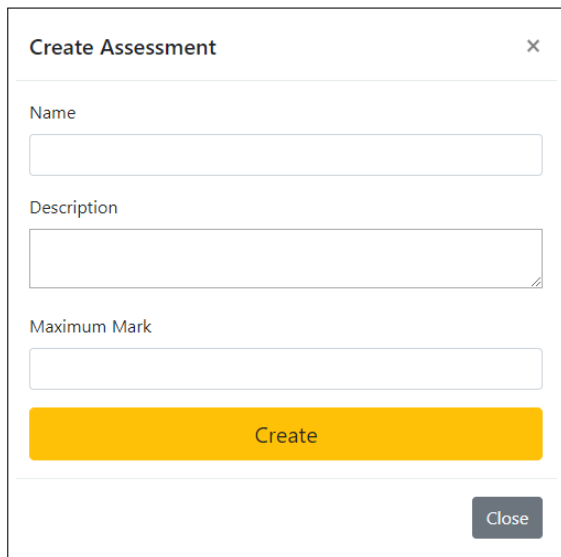
Profile Lessons Assessments Gradebook [Logout](#)

## REII414

Flashback to reality! There goes net neutrality! [×](#)

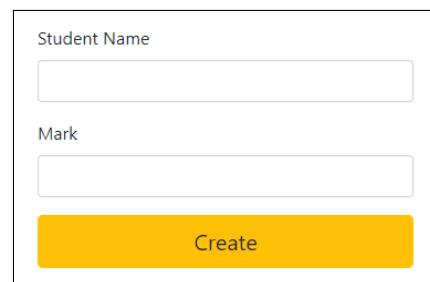
Student	Assessment Name	Mark
jpbeukes01@gmail.com	Test 1	3

Figure 4.8: Lecturer gradebook page



A modal form titled "Create Assessment" with a close button (X) in the top right corner. It contains three input fields: "Name", "Description" (with a text area icon), and "Maximum Mark". Below the fields is a large yellow "Create" button and a smaller grey "Close" button in the bottom right corner.

Figure 4.9: Creating assessments page

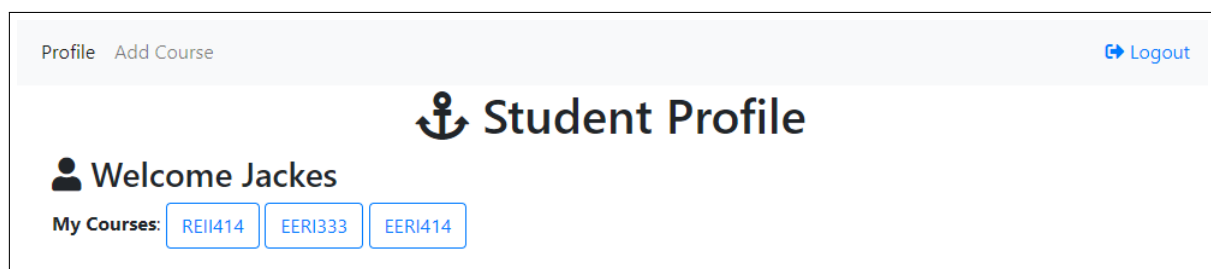


A form for updating student marks. It has two input fields: "Student Name" and "Mark". Below the fields is a large yellow "Create" button.

Figure 4.10: Update student marks page

### 4.3 Student Interface

Students have the same interface as lecturers, except for the editing functionality. No add, update or delete privileges are given to students, aside from the option of adding courses to their site. The profile page for students are given in figure 4.11, but all other examples of student pages are omitted in favour of brevity.



A student profile page. At the top, there are links for "Profile" and "Add Course", and a "Logout" button with a key icon. The main heading is "Student Profile" with an anchor icon. Below this, it says "Welcome Jackes" with a user icon. Under "My Courses:", there are three buttons labeled "REI414", "EERI333", and "EERI414".

Figure 4.11: Student profile page

## 5 Back-end

### 5.1 Sign up and Login

The 'passport' package in Nodejs allows the site to handle sessions as well as cookies in an organised way. The session stores the current user's personal info inside his/her browser to allow for a personalised site.

The 'bcrypt' package allows the passwords to be encrypted in the database with the sha256 algorithm. This ensures that the system administrators can enter into the user's sites.

Figures 5.1 and 5.2 can be summarised as, insert new user into database with encrypted password. The user logging in gets flashed error messages; should the details entered be wrong.

```
// =====  
// LOCAL SIGNUP =====  
// =====  
// we are using named strategies since we have one for login and one for signup  
// by default, if there was no name, it would just be called 'local'  
  
passport.use(  
  'local-signup',  
  new LocalStrategy({  
    // by default, local strategy uses username and password, we will override with email  
    usernameField : 'username',  
    passwordField : 'password',  
    passReqToCallback : true // allows us to pass back the entire request to the callback  
  },  
  function(req, username, password, done) {  
    // find a user whose email is the same as the forms email  
    // we are checking to see if the user trying to login already exists  
    connection.query("SELECT * FROM users WHERE username = ?",[username], function(err, rows) {  
      if (err)  
        return done(err);  
      if (rows.length) {  
        return done(null, false, req.flash('signupMessage', 'That username is already taken.'));  
      } else {  
        // if there is no user with that username  
        // create the user  
        var newUserMysql = {  
          username: username,  
          password: bcrypt.hashSync(password, null, null) // use the generateHash function in our user model  
        };  
  
        var insertQuery = "INSERT INTO users ( username, password ) values (?,?)";  
  
        connection.query(insertQuery,[newUserMysql.username, newUserMysql.password],function(err, rows) {  
          newUserMysql.id = rows.insertId;  
  
          return done(null, newUserMysql);  
        });  
      }  
    });  
  })  
);
```

Figure 5.1: Code: Local Sign-up

```
// =====
// LOCAL LOGIN =====
// =====
// we are using named strategies since we have one for login and one for signup
// by default, if there was no name, it would just be called 'local'

passport.use(
  'local-login',
  new LocalStrategy({
    // by default, local strategy uses username and password, we will override with email
    usernameField : 'username',
    passwordField : 'password',
    passReqToCallback : true // allows us to pass back the entire request to the callback
  },
  function(req, username, password, done) { // callback with email and password from our form
    connection.query("SELECT * FROM users WHERE username = ?",[username], function(err, rows){
      if (err)
        return done(err);
      if (!rows.length) {
        return done(null, false, req.flash('loginMessage', 'No user found.')); // req.flash is the way to set flashdata using connect-flash
      }

      // if the user is found but the password is wrong
      if (!bcrypt.compareSync(password, rows[0].password))
        return done(null, false, req.flash('loginMessage', 'Oops! Wrong password.')); // create the loginMessage and save it to session as flashdata

      // all is well, return successful user
      return done(null, rows[0]);
    });
  });
);
```

Figure 5.2: Code: Local Login

## 5.2 Courses

Courses can be created and edited by lecturers, whereas students can only view the courses. Figure 5.3 shows how a new course is added to the database. Figure 5.4 shows how the course and all of its content is retrieved as a JSON object from the database and rendered to the user.

```
//post add new course to db
app.post('/courses/new', function(req,res){
  // console.log(req.body);
  let sql = 'INSERT INTO courses (courseName, courseDescription, userID) VALUES (?,?,?)';
  let courseName = req.body.courseName;
  let courseDesc = req.body.courseDescription;
  let query = connection.query(sql,[courseName, courseDesc, req.user.id], (err, results) => {
    if(err) throw err;
    //add demo lesson
    let demo = 'demo';
    let sql = 'INSERT INTO lessons (COURSE_FK, lessonNumber, LESSON_NAME, LESSON_DESCRIPTION, LESSON_MATERIAL) VALUES (?,1,?,?,?)';
    let query = connection.query(sql,[results.insertId, demo,demo,demo], (err, results) => {
      if(err) throw err;
      res.redirect('/profile');
    });
  });
});
```

Figure 5.3: Code: Create new course

```
// =====  
// Individual Courses =====  
// =====  
//open individual course and it's lessons for student  
app.get('/course/:courseName', function(req,res){  
  //select course and it's corresponding lessons  
  let sql = 'SELECT * FROM lessons,courses WHERE lessons.COURSE_FK = courses.COURSE_ID and courses.courseName = ? order by lessonNumber as  
  let query = connection.query(sql,[req.params.courseName], (err, results2) => {  
    if(err) throw err;  
    if(isEmpty(results2)) {  
      if(req.user.lecturer > 0) {  
        res.redirect('/profile'); //moet course object paas maar is nie een  
      } else {  
        res.redirect('/profile');  
      };  
    } else {  
      if(req.user.lecturer > 0){  
        res.render('courseEdit.ejs',{user: req.user, course: results2});  
      } else {  
        res.render('courseView.ejs',{user: req.user, course: results2});  
      }  
    }  
  });  
});
```

Figure 5.4: Code: Get Courses

### 5.3 Lessons

The code in Figure 5.5 shows the post request when the lecturer creates a new lesson. The details of the lesson is added with the files and are sent to the server and database for later retrieval.

```
// =====  
// Lessons =====  
// =====  
app.post('/course/:courseName/addlesson', function(req, res){  
  //get course id  
  var form = new formidable.IncomingForm();  
  form.uploadDir = "/";  
  form.parse(req, function (err, fields, files) {  
    var oldpath = files.fileupload.path;  
    var newpath = createMaterialPath(req.params.courseName) + '/' + files.fileupload.name;  
    fs.rename(oldpath, newpath, function (err) {  
      if (err) throw err;  
      // res.write('File uploaded and moved!');  
      // res.end();  
    });  
  
    var lessonNumber = fields.lessonNumber;  
    var LESSON_NAME = fields.LESSON_NAME;  
    var LESSON_DESCRIPTION = fields.LESSON_DESCRIPTION;  
    var LESSON_MATERIAL = fields.LESSON_MATERIAL;  
    let sql = 'SELECT COURSE_FK FROM lessons,courses WHERE lessons.COURSE_FK = courses.COURSE_ID and courses.courseName = ?';  
    let query = connection.query(sql, [req.params.courseName], (err, results) => {  
      if(err) throw err;  
      console.log(results[0].COURSE_FK);  
      var courseid = results[0].COURSE_FK;  
  
      let sql = 'INSERT INTO lessons (COURSE_FK, lessonNumber, LESSON_NAME, LESSON_DESCRIPTION, LESSON_MATERIAL) VALUES (?, ?, ?, ?, ?)';  
      let query = connection.query(sql, [courseid, lessonNumber, LESSON_NAME, LESSON_DESCRIPTION, files.fileupload.name], (err, results) => {  
        if(err) throw err;  
        res.redirect('/course/'+req.params.courseName);  
      });  
    });  
  });  
});
```

Figure 5.5: Code: Create Lessons

## 5.4 Assessments

The lecturer can add assessments to a course and enter the marks of the students for a specific assessment. Figure 5.6 below shows how a new assessment is posted to the database and how marks are inserted for a specific user.

```

app.post('/course/:courseName/addAssessment', function(req, res){
  let sql = 'SELECT COURSE_ID FROM assessments,courses WHERE courses.courseName = ?';
  let query = connection.query(sql, [req.params.courseName], (err, results) => {
    if(err) throw err;
    // console.log(results[0].COURSE_FK);
    var courseid = results[0].COURSE_ID;

    let sql = 'INSERT INTO assessments (COURSE_FK, ASSESSMENT_NAME, ASSESSMENT_DESCRIPTION, ASSESSMENT_MAX_MARK) VALUES (?, ?, ?, ?)';
    var ASSESSMENT_NAME = req.body.ASSESSMENT_NAME;
    var ASSESSMENT_DESCRIPTION = req.body.ASSESSMENT_DESCRIPTION;
    var ASSESSMENT_MAX_MARK = req.body.ASSESSMENT_MAX_MARK;
    let query = connection.query(sql, [courseid, ASSESSMENT_NAME, ASSESSMENT_DESCRIPTION, ASSESSMENT_MAX_MARK], (err, results) => {
      if(err) throw err;
      console.log(results);
      res.redirect('/course/'+req.params.courseName);
    });
  });
});

//inserts marks for student on assessment
app.post('/course/:courseName/assessment/:ASSESSMENT_ID', function(req, res){

  let sql = 'SELECT id FROM users WHERE users.username = ?';
  let query = connection.query(sql, [req.body.username, req.params.ASSESSMENT_ID], (err, results) => {
    if(err) throw err;

    var STUDENT_FK = results[0].id; // moet verander
    var MARK_SCORE = req.body.MARK_SCORE;
    var assID = parseInt(req.params.ASSESSMENT_ID);

    let sql = 'INSERT INTO marks (STUDENT_FK, ASSESSMENT_FK, MARK_SCORE) VALUES (?, ?, ?)';
    let query = connection.query(sql, [STUDENT_FK, assID, MARK_SCORE], (err, results) => {
      if(err) throw err;
      // console.log(results);
      res.redirect('/course/'+req.params.courseName);
    });
  });
});

app.get('/course/:courseName/assessment/:ASSESSMENT_ID/updateMarks', function(req, res){
  res.render('updateMarks.ejs', {courseName: req.params.courseName, assessmentID:req.params.ASSESSMENT_ID});
});

```

Figure 5.6: Code: Create and Update Marks for Assessments

## 5.5 Grade-book

Lastly the grade-book is split into two types, one for students and one for lecturers. The student grade book only gets the student's mark for all the assessments in that course. The lecturer's grade book show all the students and their respective mark for all of the assessments.

```
// =====
// =====Gradebook=====
// =====
//vir students view
app.get('/course/:courseName/gradebook', function(req, res){
  if(req.user.lecturer > 0){
    let sql = 'SELECT * FROM users,courses,assessments,marks WHERE assessments.COURSE_FK = courses.COURSE_ID and marks.ASSESSMENT_FK =
assessments.ASSESSMENT_ID and marks.STUDENT_FK = users.id and courses.courseName = ? order by ASSESSMENT_NAME asc';
    let query = connection.query(sql, [req.params.courseName], (err, results) => {
      if(err) throw err;
      // console.log(results);
      req.flash('info', 'Flashback to reality! There goes net neutrality!');
      // res.json(results);
      res.render('gradebook.ejs', {message: req.flash('info'), marks: results, courseName:req.params.courseName}); //get marks for user
    });
  } else {
    let sql = 'SELECT * FROM marks,users,assessments,courses WHERE marks.STUDENT_FK=users.id and
marks.ASSESSMENT_FK=assessments.ASSESSMENT_ID and assessments.COURSE_FK = courses.COURSE_ID and users.id = ? and
courses.courseName =?';
    let query = connection.query(sql, [req.user.id, req.params.courseName], (err, results) => {
      if(err) throw err;
      // console.log(results);
      req.flash('info', 'Flashback to reality! There goes net neutrality!');
      // res.json(results);
      res.render('gradebook-student.ejs', {message: req.flash('info'), marks: results, courseName:req.params.courseName}); //get marks
for user
    });
  }
});
});
```

Figure 5.7: Code: Grade book

## 6 Conclusion

### 6.1 Strenghts

### 6.2 Flaws

### 6.3 Improvements

Future versions of the website could include a section for online assessments, where lecturers are able to create new tests and students can complete existing tests. The website must then be able to evaluate the student's answers and update their marks.

The current version of the website can only be accessed via a local network. To launch the site for global use, a domain name have to be acquired.



## 6.4 Techniques Learned

HTTP post and get requests were used extensively in this project. Hence, much knowledge of the protocol and its header structure was acquired.