# Lab Exercise: jQuery and AJAX

Overview of the lab exercise:

- Developing an AJAX script in to validate email availability during registration.
- Implementing routes to handle email checking and user registration processes.

1. In the existing database, create a table with the following structure:
    - Table name: users
    - Attributes:
        - id (integer, primary key, auto-increment)
        - username (varchar, 50)
        - email (varchar, 50)
        - password (varchar, 50)

2. Create the Pug view:

    In your views directory, create a file register.pug.

```
doctype html
html
  head
    title User Registration
    script(src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js")
    link(href="/stylesheets/register.css", rel="stylesheet")
  body
    form#register-form(method='post')
      h1 User Registration
      label(for='name') Name:
      input#name(type='text', name='name', required)
      label(for='email') Email:
      input#email(type='email', name='email', required)
      label(for='password') Password:
      input#password(type='password', name='password', required)
      button#register-button(type='submit') Register
      button#register-button(type='reset') Reset
      p#email-error
    script(src='/javascripts/checkEmail.js')
```

3. Create the AJAX script:

   In your public directory, create a file checkEmail.js.

```
$(document).ready(function() {

    $('#email').on('input', function() {

      var email = $('#email').val();

      $.ajax({

        url: '/checkEmail',

        data: { email: email },

        success: function(data) {

          if (data.exists) {

            $('#register-button').prop('disabled', true);

            $('#register-button').css({

              'background-color': '#ff7f6e',

              'color': 'black'

            });

            $('#email').css('background-color', '#ffc0b8');

            $('#email-error').text('Email already registered!').css('color', 'red');

          } else {

            $('#register-button').prop('disabled', false);

            $('#email').css('background-color', '#d7ffb8');

            $('#register-button').css({

              'background-color': '',

              'color': ''

            });

            $('#email-error').text('Email is available!').css('color', 'green');

          }

        }

      });

    });

  });
```

4. Create a new file named email.js inside the routes folder of your project. This file will contain the route that checks if an email entered by the user in the registration form already exists in the database.

5. In email.js, define a route that responds to GET requests at the /checkEmail endpoint.

```
module.exports = function(db) {
    var express = require('express');
    var router = express.Router();

    router.get('/checkEmail', function(req, res) {
      var email = req.query.email;
      db.connect(function(err) {
        if (err) throw err;
        db.query("SELECT * FROM users WHERE user_email = ?", [email], function (err,
result) {
          if (err) throw err;
          var exists = result.length > 0;
          res.json({ exists: exists });
        });
      });
    });
    return router;
  }
```

module.exports = function(db) {...}: Exports a function that takes a db object as an argument. This db object is used to interact with your database.

var express = require('express'); var router = express.Router();: Import the Express.js module and create a new router object.

router.get('/checkEmail', function(req, res) {...});: Defines a new route that listens for GET requests at the /checkEmail endpoint.

var email = req.query.email;): Retrieve the email from the query parameters

db.query("SELECT * FROM users WHERE user_email = ?", [email], function (err, result) {...});: Connect to the database and execute a SQL query to find a user with the given email

result.length > 0: If a user with the given email is found, set exists to true. Otherwise, exists is false.

res.json({ exists: exists });: Send a JSON response back to the client with the exists value

6. After defining the route, make sure to use it in app.js by requiring the email.js file and using it as middleware:

```
var checkEmailRoute = require('./routes/email')(db);
app.use('/', checkEmailRoute);
```

7. Create a new file named register.js inside the routes folder of your project. This file will contain the route that will handle the registration process.

8. In register.js, define two routes - a GET route to display the registration form and a POST route to process the form data.

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res) {
  res.render('register');
});

router.post('/', function(req, res) {
  var name = req.body.name;
  var email = req.body.email;
  var password = req.body.password;
  const query = 'INSERT INTO users (user_name, user_email, user_password) VALUES (?, ?, ?)';
  req.db.query(query, [name, email, password], (err, results) => {
    if (err) throw err;
    res.redirect('/login');
  });
});

module.exports = router;
```

9. Require the new route file in app.js file and use it. Add a middleware function that attaches the db object to the req object:

```
app.use(function(req, res, next) {
  req.db = db;
  next();
});
var registerRoutes = require('./routes/register');
app.use('/register', registerRoutes);
```