Further Web Design & Development

(CT117-3-2)

Individual Assignment

Name:	**Omar Mahmoud Elkady**
ID:	**TP068703**
Weightage:	**100%**

Project Title: **Pyledge (Python Knowledge)**

Table of Contents

Introduction

Welcome to Pyledge, a comprehensive educational platform focused on improving the learning experience for both students and instructors. Pyledge strives to create a fascinating and user-friendly environment that provides a smooth learning experience and efficient user management. Whether you're a student looking to expand your Python knowledge or a teacher looking to improve educational outcomes, Pyledge has the tools and resources to offer to both the student as well as the instructor.

Objectives: -

1. **Improve Learning**: Give users a variety of learning tools, such as books and exercises, to assist them master Python programming.
2. **User Management**: Ensure effective user management, including authentication and profile management.
3. **Interactive Learning**: Use exercises to reinforce understanding.
4. **Resource Accessibility**: Make resources such as books readily available and downloadable to users.

Scope: -

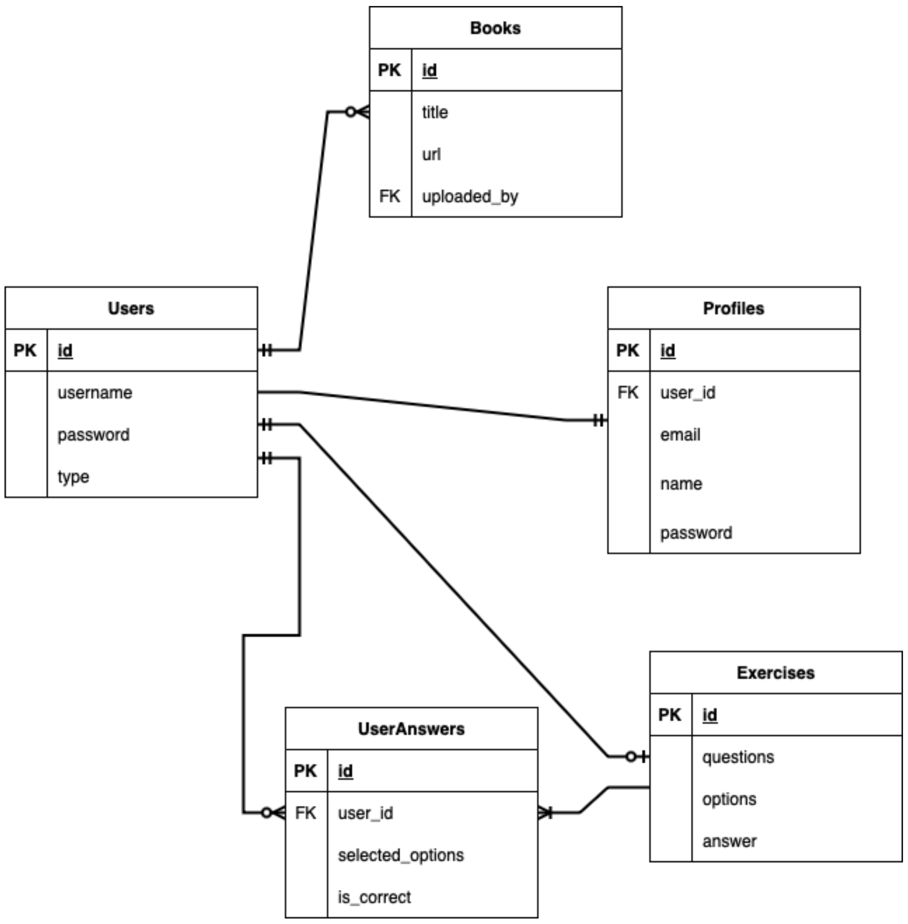
1. **User Authentication and Management**:
   - User signup and login.
   - Profile management, including modification and deletion.
   - Security features include session control and time-outs.

2. **Content Management**:
   - Providing access to a library of Python programming books.
   - Allows users to download content.

3. **Interactive Exercises**:
   - Providing tasks and quizzes to assess users understanding.
4. **Dashboard**:
   - A primary hub that allows users to travel through various sections of the website.
   - Displays website-specific data and introduction.

Design

1. **ERD**



The Entity-Relationship Diagram (ERD) for Pyledge depicts the database structure, emphasising the major entities, their attributes, and the links between them.

1. **Users**: This object maintains user information such as their unique identification, username, email address, password, and user type (student or teacher).

2. **Profile**: Each user has a profile with a unique profile identifier that is linked to the user's ID. The profile includes the user's name, email address, and password.

3. **Books**: This object controls the books available on the platform, providing a unique book identification, title, access URL, and user identifier.

4. **Exercises**: This entity comprises exercises with a unique exercise identity, question text and the correct answer.

5. **UserAnswers**: This object stores the responses entered by users for exercises. It contains a unique answer identifier, the user's identifier, the exercise's identifier, the options chosen by the user, and whether the selected option is correct.

Relationships:

Users to Profiles: one-to-one (each user has their own profile).

Users to Books: One-to-Many (each user may download numerous books).

Users to UserAnswers: One-to-Many (each user can provide many answers).

Exercises to User Answers: One-to-Many.

2. Data Dictionary

2.1 Users

Column	Data Type	Constraints	Description
id	INT	PK, Auto Increment	Unique identifier for each user
username	VARCHAR(50)	NOT NULL	Username of the user
email	VARCHAR(100)	NOT NULL	Email address of the user
password	VARCHAR(255)	NOT NULL	Password for authentication
type	ENUM('student', 'instructor'	NOT NULL	Type of user

2.2 Profile

Column	Data Type	Constraints	Description
id	INT	PK, Auto Increment	Unique identifier for each profile
User_id	INT	FK, NOT NULL	
name	VARCHAR(100)	NOT NULL	Name of user
email	VARCHAR(100)	NOT NULL, Unique	Email of user
password	VARCHAR(255)	NOT NULL	Password for authentication

2.3 Books

Column	Data Type	Constraints	Description
id	INT	PK, Auto Increment	Unique identifier for each book

title	VARCHAR(255)	NOT NULL	Title of the book
url	VARCHAR(255)	NOT NULL	URL to access book
Uploaded_by	INT	FK, NOT NULL	

2.4 Exercise

Column	Data Type	Constraints	Description
id	INT	PK, Auto Increment	Unique identifier for exercise
question	TEXT	NOT NULL	Question text
options	JSON	NOT NULL	JSON array
answer	VARCHAR(255)	NOT NULL	Correct Answer

2.5 UserAnswers

Column	Data Type	Constraints	Description
id	INT	PK, Auto Increment	Unique identifier for answer
User_id	INT	FK, NOT NULL	
exercise_id	INT	FK, NOT NULL	
selected_options	VARCHAR(255)	NOT NULL	The options selected by user
is_correct	BOOLEAN	NOT NULL	Indicates if selected option is correct

1. **Users**

This table contains information about all platform users, such as their unique identities, login credentials, and role (student or instructor).

2. **Profile**

This table contains detailed profile information for users, tying each profile to a specific user and keeping personal information such as name, email, and password.

3. **Books**

This table keeps track of the books available on the platform, including their names, access URLs, and the identities of the individuals that uploaded them.

4. **Exercise**

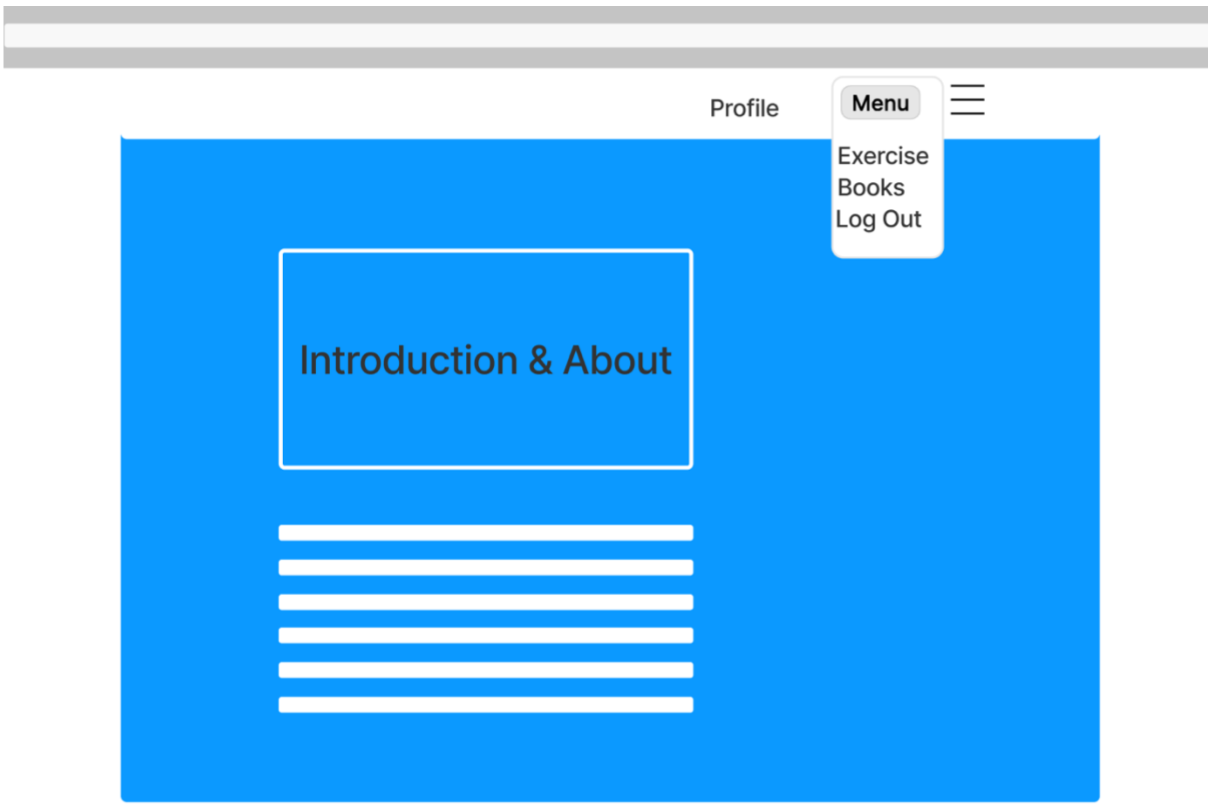
This table provides exercises that users can perform, including questions, possible answers, and correct answers.

5. **User Answers**

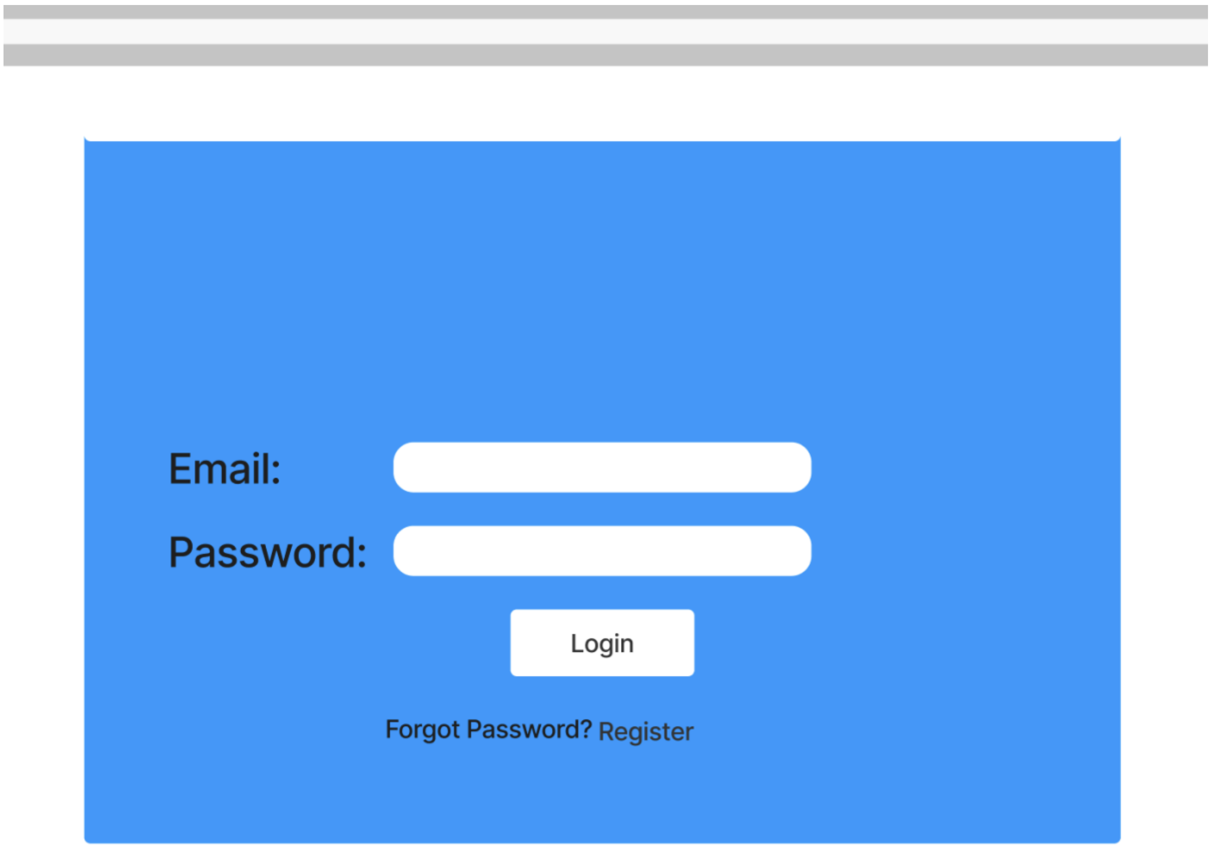
This table keeps track of users responses to exercises, assigning each response to a specific user and exercise, and marking if the user's chosen answer is valid.

3. Wireframes

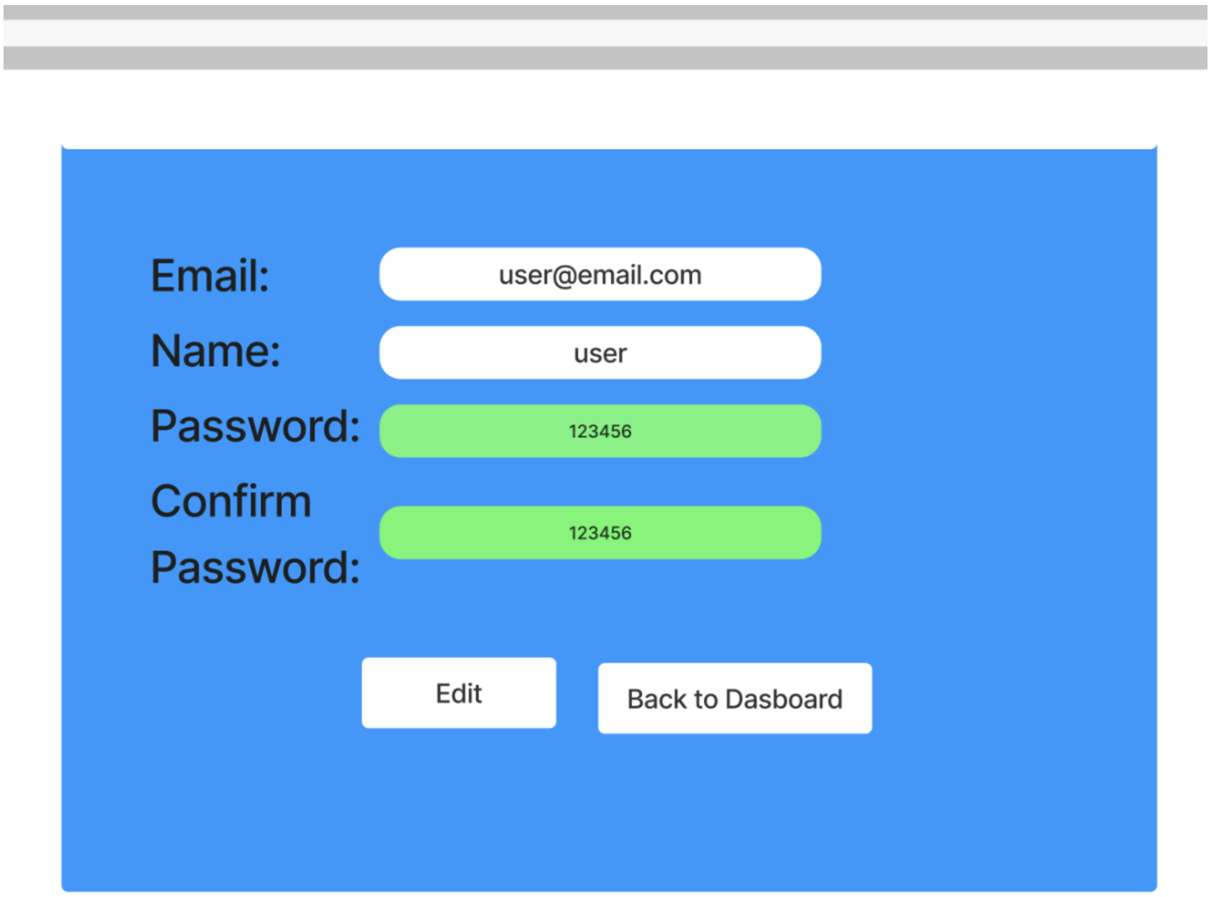
a. Dashboard



b. Login Page



c. Profile Page

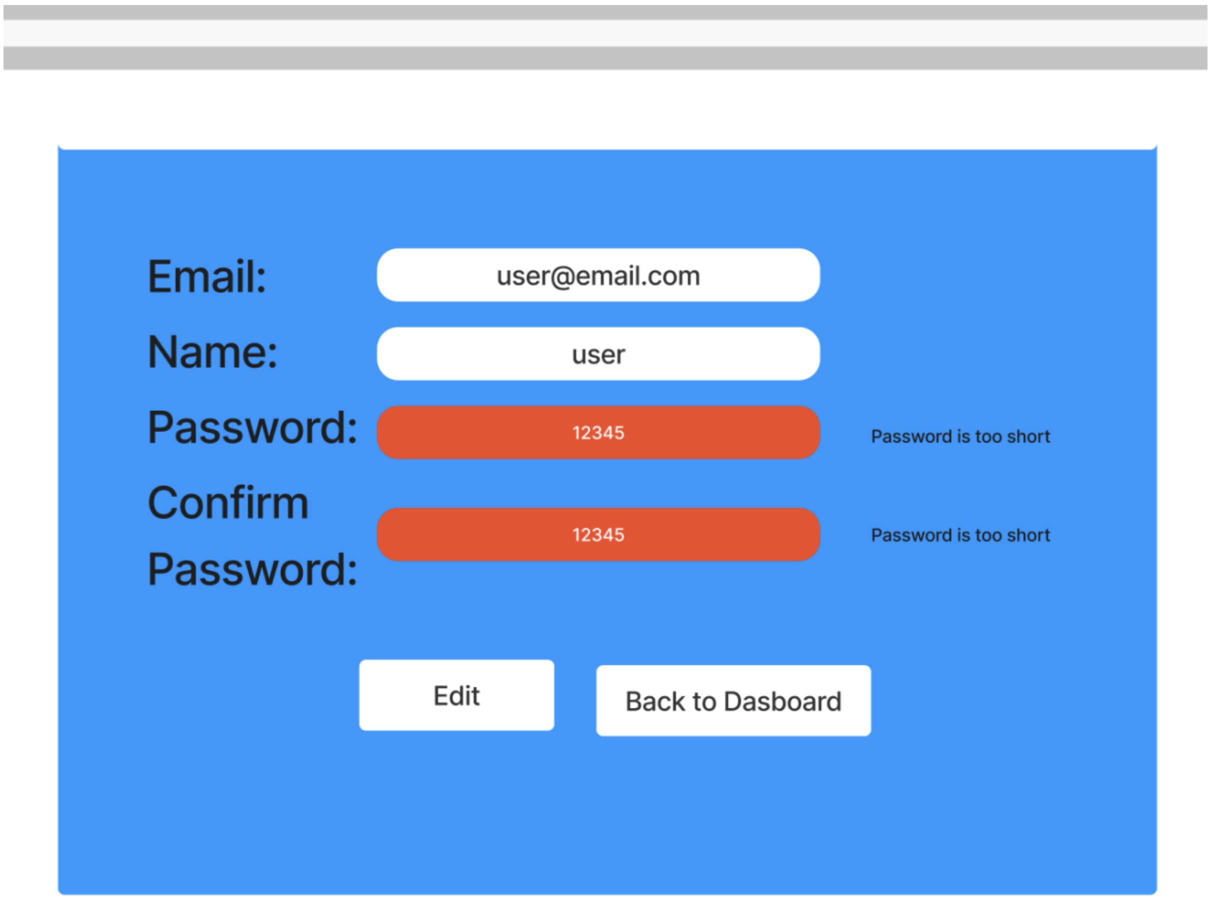


Email: user@email.com

Name: user

Password: 123456

Confirm Password: 123456

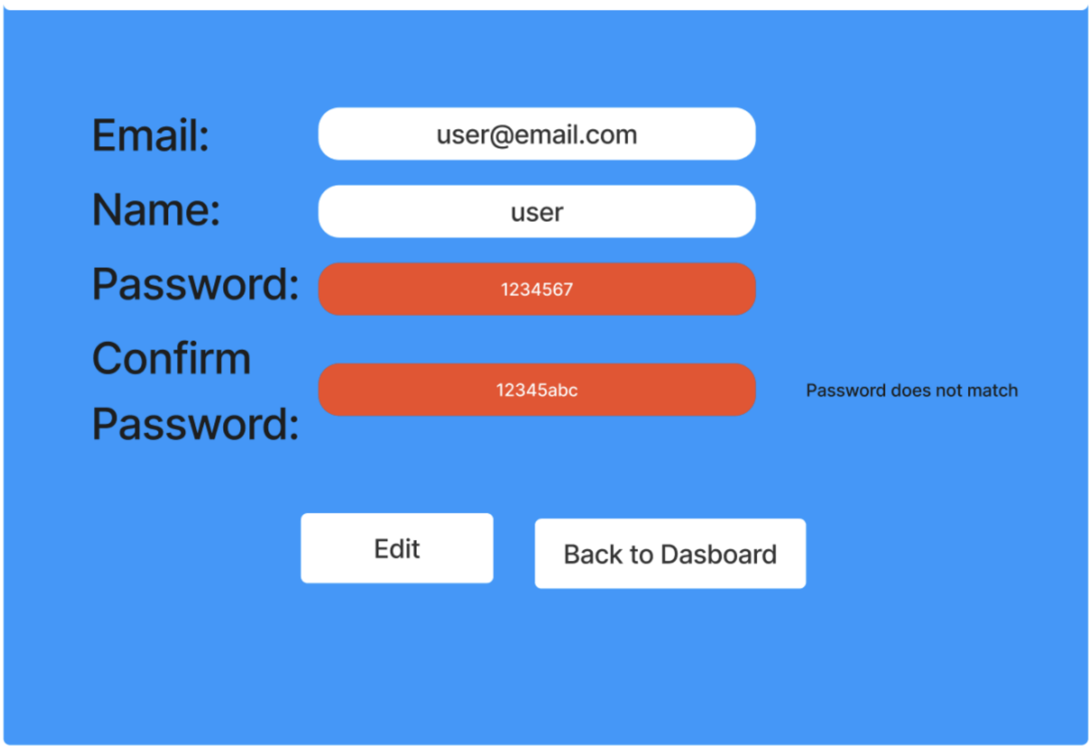


Email: user@email.com

Name: user

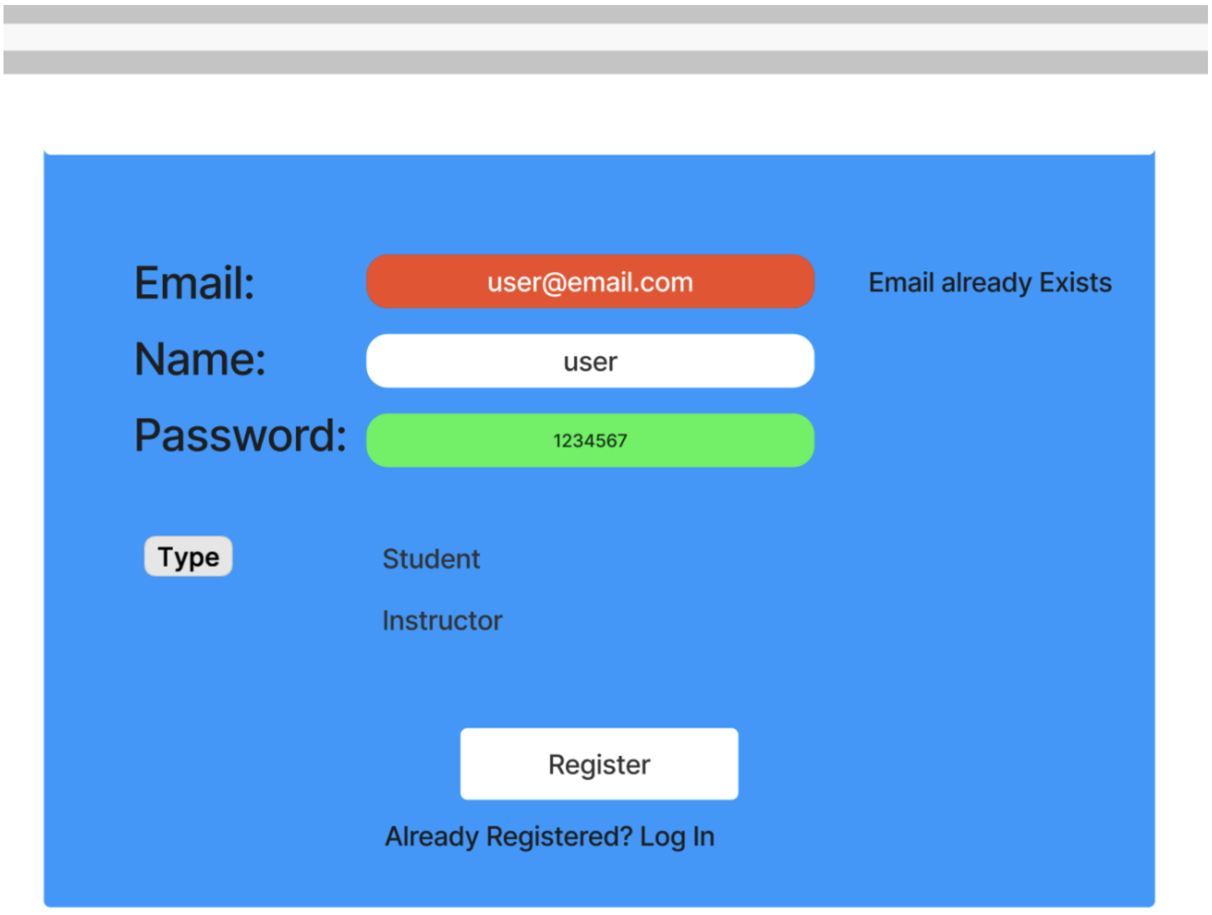
Password: 12345 Password is too short

Confirm Password: 12345 Password is too short



Email:	user@email.com	
Name:	user	
Password:	1234567	
Confirm Password:	12345abc	Password does not match

d. Register



Email: user@email.com Email already Exists

Name: user

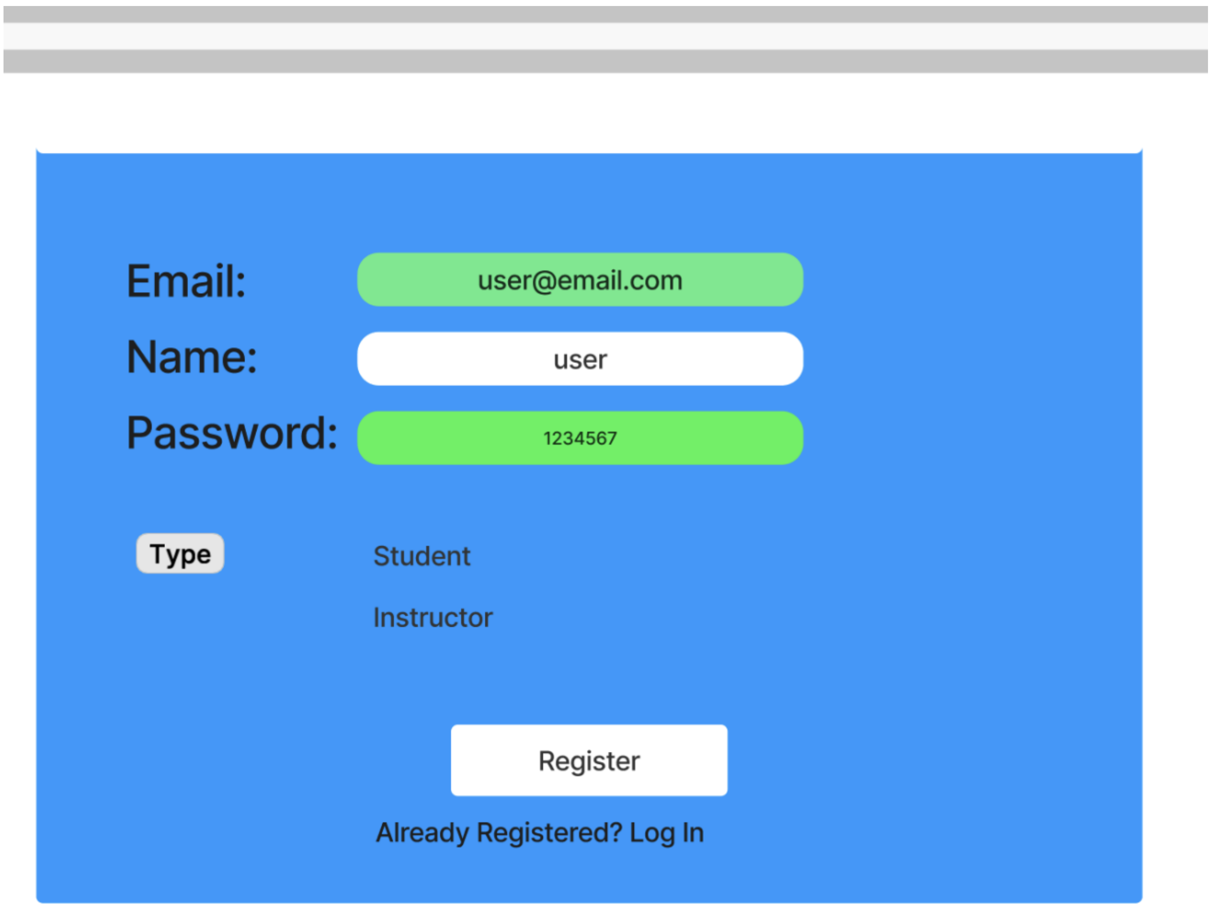
Password: 1234567

Type Student

Instructor

Register

Already Registered? Log In



Email: user@email.com

Name: user

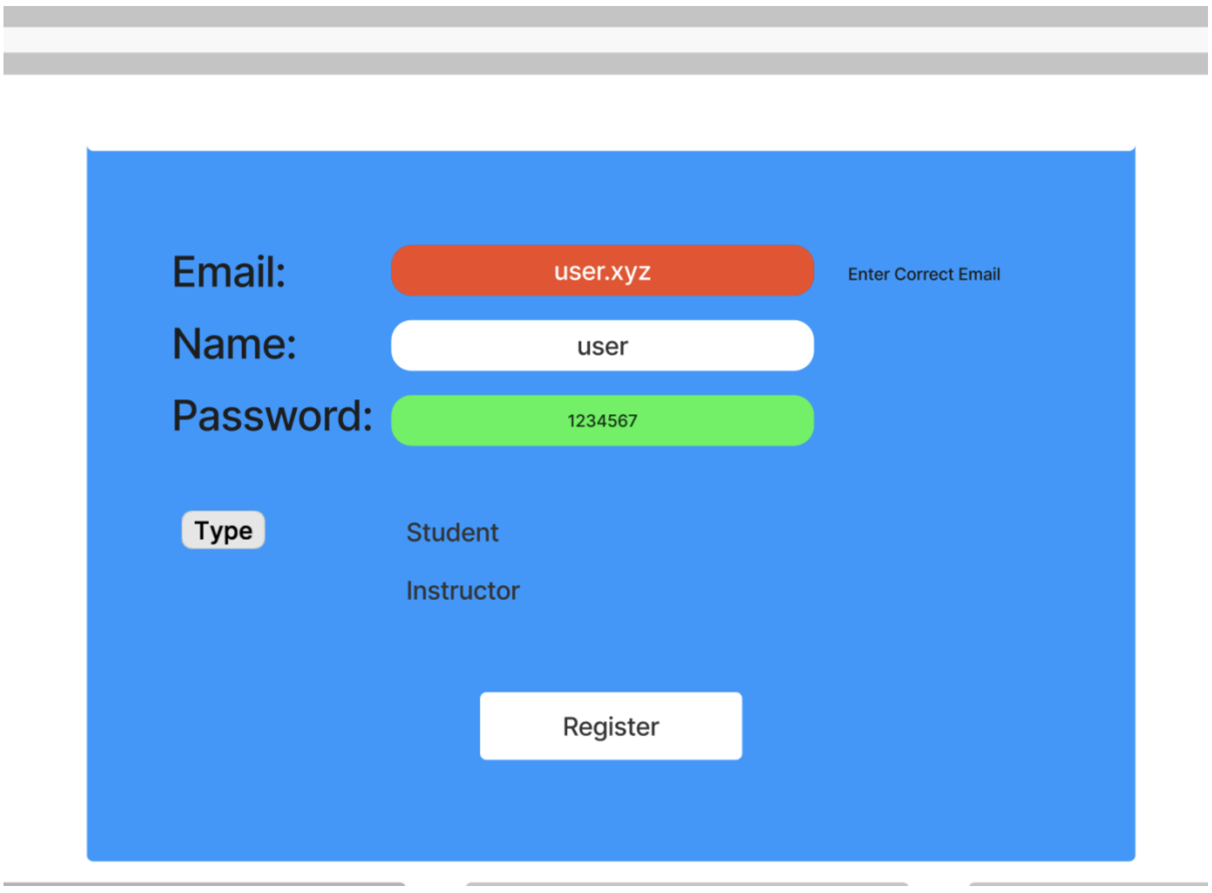
Password: 1234567

Type Student

Instructor

Register

Already Registered? Log In



Description of the provided wireframes

1. **Dashboard**

The dashboard has a "Introduction & About" section with a menu that includes access to Exercises, Books, and Log Out. The "Profile" link is likewise located at the top.

2. **Login page**

The login page contains areas for email and password, as well as a "Login" button. There are other options to "Forgot Password?" and "Register."

3. **Edit the Profile Page**

The edit profile page allows users to change their email address, name, and password. It contains password validation feedback and a button for confirming updates or returning to the Dashboard.

4. **Password Validation: Too Short**

This edit profile page displays validation problems, indicating that the password is too short. The password and confirmation password fields are

5. **Password Validation: Does Not Match**

This edit profile page shows that the passwords do not match. The password and confirm password areas are marked in red, and any mismatched passwords generate an error notice.

6. **Registration Page: Email Already Exists**

The registration screen provides fields for email, name, and password, with feedback showing that the email address is already in the system. There is a drop-down box to choose the user type (Student or Instructor) and a "Register" button.

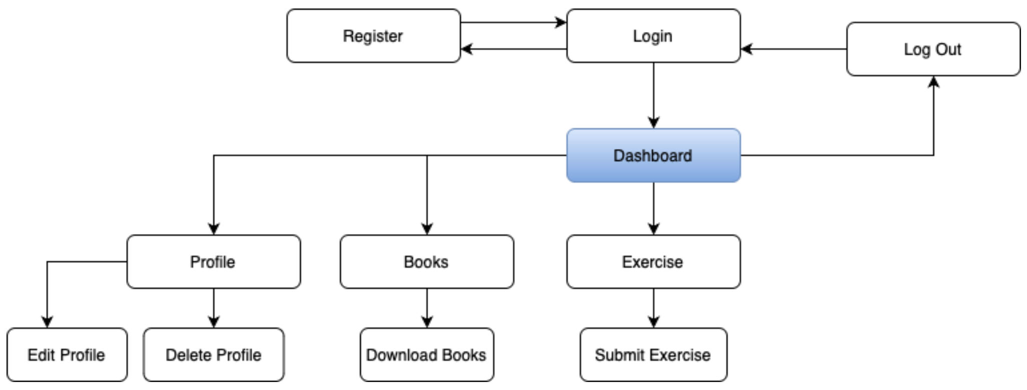
7. **Registration Page: Valid Input**

This signup page displays proper input fields, with the email, name, and password correctly entered. The user type selection and "Register" button are also available.

8. **Registration Page: Invalid Email**

The registration page displays an invalid email format.

4. Website Navigational Structure



This figure depicts the navigation path for the Pyledge educational platform:

Register: Users can create a new account, which will allow them to login.

Login: Users must log in to access the platform. A successful login takes users to the Dashboard.

Dashboard: The central hub of the application, offering access to the following sections

Profile: Users can see, modify, and remove their profiles.

Edit Profile: Users can alter their personal information.

Delete Profile: Users can remove their profiles from the platform.

Books: Users can read and download instructive books.

Exercise: Users can access exercises and submit their responses.

Submit Exercise: Users enter their responses to exercises.

Log Out: Users can log out from any point in the programme and return to the login page

Code Snippet Explanation (Implementation)

1. JavaScripts

a. checkEmail

```
$(document).ready(function() {
  $('#email').on('input', function() {
    var email = $('#email').val();
    console.log('Checking email:', email);
    $.ajax({
      url: '/checkEmail',
      data: { email: email },
      success: function(data) {
        console.log('Response received:', 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');
        }
      },
      error: function(err) {
        console.log('AJAX error:', err);
      }
    });
  });
});
```

When registering this code snippet is executed an AJAX request is held to the server endpoint /checkEmail, which sends the current email address for validity, .if the email is already registered (data.exists = true): The registration button is disabled and designed to signal an error and the email field is marked in red. An error warning appears next to the email input area. If the email is available (data.exists = false): The register button is activated and reset to its default appearance. The email input field is highlighted green. A success message appears next to the email input area. The major goal of this code is to notify users immediately about the availability of the email address they entered during registration. This improves user experience by avoiding submissions.

b. checkPassword

```

$(document).ready(function() {
   // Password validation
   $('#password, #confirmPassword').on('input', function() {
     var password = $('#password').val();
     var confirmPassword = $('#confirmPassword').val();

     if (password.length < 6) {
       $('#password').css('background-color', '#ffc0b8');
       $('#password-error').text('Password must be at least 6 characters!').css('color', 'red');
     } else {
       $('#password').css('background-color', '#d7ffb8');
       $('#password-error').text('');
     }

     if (password !== confirmPassword) {
       $('#confirmPassword').css('background-color', '#ffc0b8');
       $('#confirmPassword-error').text('Passwords do not match!').css('color', 'red');
     } else if (confirmPassword.length >= 6) {
       $('#confirmPassword').css('background-color', '#d7ffb8');
       $('#confirmPassword-error').text('Passwords match!').css('color', 'green');
     } else {
       $('#confirmPassword').css('background-color', '');
       $('#confirmPassword-error').text('');
     }
   });
 });
```

$('#password, #confirmPassword').on('input', function() This creates an event listener for the password and confirm password input fields, triggering anytime the user types or changes the information. If the password is less than 6 characters long, the password input field's background colour is red (#ffc0b8), and an error warning is displayed, stating that the password must be at least 6 characters long.

If the password is at least 6 characters long, the password input field's background colour is green (#d7ffb8), any prior error messages are cleared. If the passwords do not match, the confirm password input field's background colour is red (#ffc0b8), and an error notice appears suggesting that the passwords do not match.

The major goal of this code is to validate password fields in real time, giving users immediate feedback on the length of their password and whether the password and confirm password fields match. This ensures that users provide a valid and matching password, hence improving the overall user experience and eliminating errors during form submission.

2. Routes

a. register

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

module.exports = (db) => {

  router.post('/register', (req, res) => {
    const { email, username, password, type } = req.body;

    if (password.length < 6) {
        return res.status(400).json({ error: 'Password must be at least 6 characters long' });
    }
    const query = 'INSERT INTO users (email, username, password, type) VALUES (?, ?, ?, ?)';

    db.query(query, [email, username, password, type], (err, results) => {
      if (err)throw err;
      res.redirect('/login')
    });
  });

```

```
    return router;
};

```

After POST route has been defined for /register, the route handler function retrieves the email address, username, password, and type from the request body. The password's length is checked, if the password is less than six characters long, a 400 status code and an error notice are returned. An SQL query string is then used to create a new user record in the users table. The db.query function is used to execute the query, which takes the email, username, password, and type as inputs. If an error occurs while running the query, it is thrown.
Following successful execution, the user is redirected to the /login page. The major function of this code is to manage the user registration procedure. It validates the password length, creates a new user record in the database, and redirects the user to the login page after successful registration. This ensures that only valid users may register and that their information is properly kept in the database.

b. **profile**

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

  // Profile page
  router.get('/profile', (req, res) => {
    if (!req.session.user) {
      return res.redirect('/login');
    }

    const user = {
      email: req.session.email,
      name: req.session.username,
      type: req.session.type
    };

    res.render('profileViews', { user });
  });
```

```
// Edit profile page
router.get('/profile/edit', (req, res) => {
    if (!req.session.user) {
        return res.redirect('/login');
    }

    const user = {
        email: req.session.email,
        name: req.session.username,
        type: req.session.type
    };

    res.render('editViews', { user });
});

// Handle profile update
router.post('/profile/edit', (req, res) => {
    if (!req.session.user) {
        return res.redirect('/login');
    }

    const { email, name, password, confirmPassword } = req.body;

    if (password !== confirmPassword) {
        return res.send('Passwords do not match.');
    }

    const updateQuery = 'UPDATE users SET email = ?, username = ?, password = ? WHERE id = ?';
    db.query(updateQuery, [email, name, password, req.session.user.id], (err, result) => {
        if (err) {
            return res.send('An error occurred.');
        }

        req.session.email = email;
        req.session.username = name;
        res.redirect('/profile');
    });
});

// Handle profile deletion
```

```
  router.post('/profile/delete', (req, res) => {
    if (!req.session.user) {
      return res.redirect('/login');
    }

    const deleteQuery = 'DELETE FROM users WHERE id = ?';
    db.query(deleteQuery, [req.session.user.id], (err, result) => {
      if (err) {
        return res.send('An error occurred.');
      }

      req.session.destroy((err) => {
        if (err) {
          return res.send('An error occurred while logging out.');
        }
        res.redirect('/login');
      });
    });
  });

  return router;
};
```

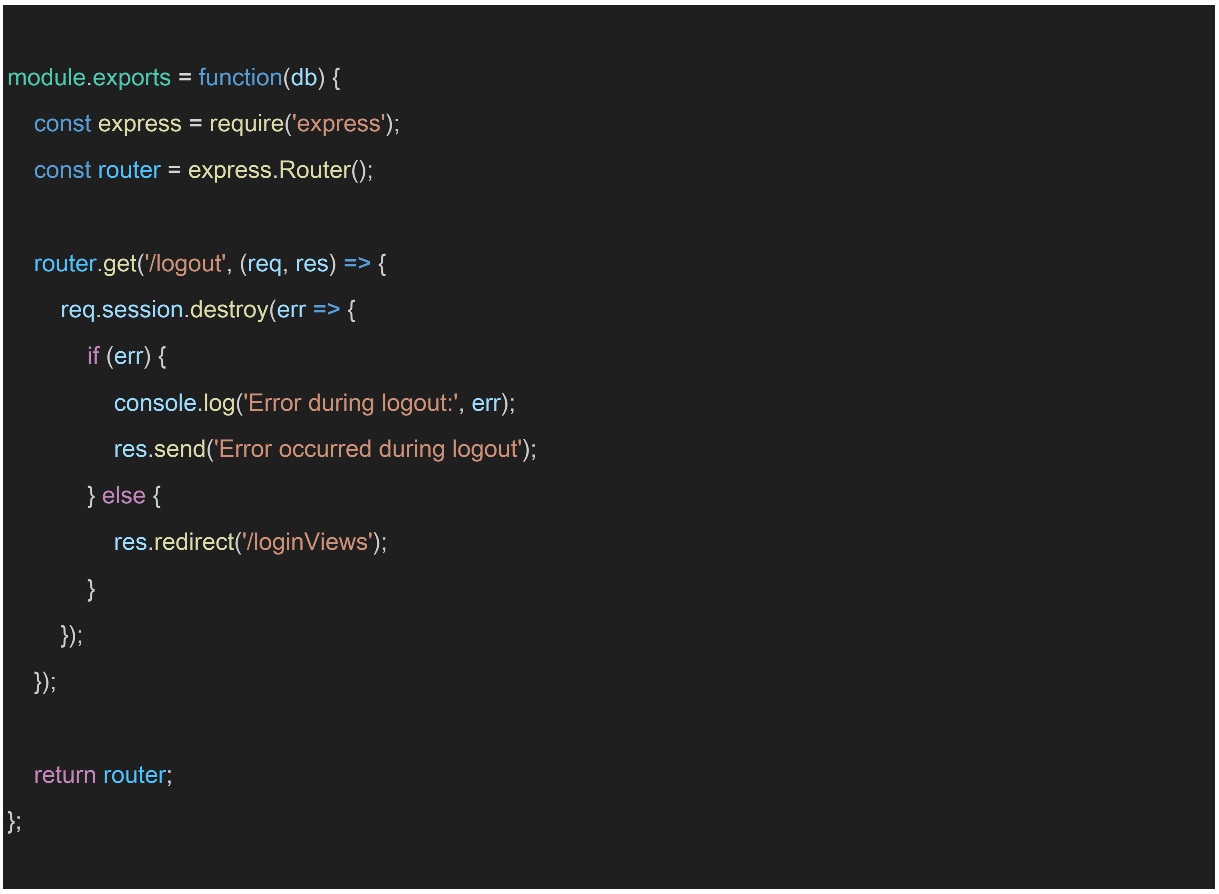
The /profile route leads to the profile page. It determines whether the user is logged in by checking the presence of req.session.user. If the user is not logged in, they will be sent to the login page. If the user is logged in, the profileViews template receives the user's session information, such as email, name, and type, and renders it.

The /profile/edit link leads to the profile editing page. Similar to the profile page route, it looks for a current user session and redirects to the login page if none is found. The editViews template receives the user's session information and pre-fills the form fields.

The /profile/edit route accepts POST requests to update the profile. It first checks for an active user session before retrieving the updated user information from the request body. If the new password and the confirmation password do not match, an error message is displayed. If the passwords match, a SQL UPDATE query is used to update the user's information in the database. After a successful update, the session information is changed and the user is sent to their profile page.

The /profile/delete route is for profile deletion. It checks the user session before executing a SQL DELETE query to remove the user from the database. If the deletion is successful, the user's session is terminated and they are routed to the login page.

c. **logout**



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

    router.get('/logout', (req, res) => {
        req.session.destroy(err => {
            if (err) {
                console.log('Error during logout:', err);
                res.send('Error occurred during logout');
            } else {
                res.redirect('/loginViews');
            }
        });
    });

    return router;
};
```

This code creates an Express.js route that handles user logout functionality in a web application. The module exports a function that accepts a db parameter, which represents the database connection, but it is not used in this particular route. Inside the function, an Express router object is built to specify the /logout route.

When a GET request is made to the /logout route, the server attempts to terminate the user's session by calling req.session.destroy(). This approach removes all session data from the server, thereby logging the user out. The method accepts a callback function to handle the session's destruction result. If the session is successfully terminated, the user is routed to the /loginViews page. This often takes the user to the login screen, where they are logged out and must authenticate again to access protected routes.

d. **exercise**

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

const questions = [
 {
  question: 'What is the output of print(2 ** 3)?',
  options: ['5', '6', '7', '8'],
  answer: '8'
 },
 {
  question: 'What does the following code do: list1 = [1, 2, 3]; list2 = list1; list1.append(4)?',
  options: ['list2 is [1, 2, 3, 4]', 'list1 is [1, 2, 3]', 'list2 is [4]', 'list1 is [4]'],
  answer: 'list2 is [1, 2, 3, 4]'
 },
 {
  question: "What is the output of print('Hello World'[::-1])?",
  options: ["dlroW olleH", "Hello World", "World Hello", "dlroW olleH "],
  answer: "dlroW olleH"
 },
 {
  question: "Which of the following is a mutable data type in Python?",
  options: ["tuple", "str", "list", "int"],
  answer: "list"
 },
 {
  question: "Which keyword is used for function in Python?",
  options: ["func", "def", "function", "define"],
  answer: "def"
 },
 {
  question: "What does the // operator do in Python?",
    options: ["Addition", "Division", "Integer Division", "Exponentiation"],
    answer: "Integer Division"
  },
  {
    question: "Which function is used to find the length of a string in Python?",
    options: ["len()", "length()", "size()", "strlen()"],
    answer: "len()"
  },
  {
    question: "What is the output of print(type(10))?",
    options: ["<class 'float'>", "<class 'str'>", "<class 'int'>", "<class 'list'>"],
    answer: "<class 'int'>"
  },
  {
    question: "Which keyword is used to create a class in Python?",
    options: ["class", "def", "function", "object"],
    answer: "class"
  },
  {
    question: "What is the correct file extension for Python files?",
    options: [".python", ".pyth", ".py", ".pt"],
    answer: ".py"
  }
];

router.get('/exercises', (req, res) => {
  if (!req.session.user) {
    return res.redirect('/loginViews');
  }

  // Shuffle and select 5 random questions
  const shuffledQuestions = questions.sort(() => 0.5 - Math.random());
  const selectedQuestions = shuffledQuestions.slice(0, 5);

  res.render('exerciseViews', { questions: selectedQuestions });
});

router.post('/exercise/submit', (req, res) => {
  const userAnswers = req.body;
  let score = 0;
  const selectedQuestions = req.session.selectedQuestions || [];

  selectedQuestions.forEach((question, index) => {
    const userAnswer = userAnswers[`question-${index}`];
    console.log(`Question ${index + 1}: ${question.question}`);
    console.log(`User Answer: ${userAnswer}`);
    console.log(`Correct Answer: ${question.answer}`);
    if (userAnswer && userAnswer === question.answer) {
      score += 2; // Each question is worth 2 marks
    }
  });

  console.log(`Final Score: ${score} out of 10`);
  res.json({ score, total: 10 });
});

module.exports = router;
```

The router.get('/exercises') route handler is used to display the quiz questions to the user. It initially determines whether the user is logged in by checking the existence of req.session.user. If the user is not logged in, they will be sent to the login page. If the user is logged in, the questions array is jumbled to create a random order, and five questions are chosen from the shuffle. These selected questions are then shown using the exerciseViews template.

The router.post('/exercise/submit') route handler handles the user's provided responses. It extracts the responses from the request body and sets the score variable. The chosen questions are then drawn from the session where for each question, the user's answer is compared to the correct answer, and if they are the same, the score is increased by two points. The final result is then recorded on the console and returned to the client as a JSON response, along with the maximum potential score of 10.

3. App.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
const session = require('express-session');
const bodyParser = require('body-parser');
const mysql = require('mysql2');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// database connection
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'omar567890',
  database: "pyledge",
});

db.connect((err) => {
  if (err) {
    console.error('Database connection failed:', err);
  } else {
    console.log('Database connection successful');
  }
});

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use(express.static(path.join(__dirname, 'public')));
app.use(session({
  secret: 'fwdd',
  resave: false,
  saveUninitialized: true,
  cookie: { secure: false }
}));

app.use(function(req, res, next) {
  req.db = db;
  next();
});

// Route registration
app.use('/', indexRouter);
app.use('/users', usersRouter);

const registerRoutes = require('./routes/register')(db);
app.use('/', registerRoutes);

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

const dashboardRoute = require('./routes/dashboard')(db);
app.use('/', dashboardRoute);

const booksRoute = require('./routes/books')(db);
app.use('/', booksRoute);
console.log('Books route included');

const profileRoute = require('./routes/profile')(db);
app.use('/', profileRoute);

const exerciseRoute = require('./routes/exercise');
app.use('/', exerciseRoute);

const logoutRoute = require('./routes/logout')(db);
app.use('/', logoutRoute);

// Login Route
```

```
app.get('/login', (req, res) => {
  res.render('loginViews'); // Ensure this matches your Pug template name
});

app.post('/login', (req, res) => {
  // Debugging: Log the received email and password
  console.log('Email:', req.body.email);
  console.log('Password:', req.body.password);

  let sql = 'SELECT * FROM users WHERE email = ? AND password = ?';
  db.query(sql, [req.body.email, req.body.password], (err, result) => {
    if (err) {
      console.error('Database query error:', err);
      res.send('An error occurred');
      return;
    }
    // Debugging: Log the result of the query
    console.log('Query Result:', result);

    if (result.length > 0) {
      req.session.user = result[0];
      req.session.email = result[0].email;
      req.session.username = result[0].username;
      req.session.type = result[0].type;
      res.redirect('/dashboard');
    } else {
      res.send('Login failed');
    }
  });
});

app.get('/registerViews', (req, res) => {
  res.render('registerViews');
});

app.get('/loginViews', (req, res) => {
  res.render('loginViews');
});

app.get('/dashboard', (req, res) => {
  if (!req.session.user) {
    return res.redirect('/loginViews');
  }
  res.render('dashboard', { username: req.session.user.username });
});


// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

The code starts by importing the required modules, which include Express for server setup, path for file system path manipulations, cookie-parser for cookie handling, morgan for HTTP request logging, express-session for session management, body-parser for parsing request bodies, and mysql2 for database interactions. This comprehensive set of imports serves as the foundation for a well-rounded application that can quickly manage user interactions, sessions, and data storage.

The application then connects to a MySQL database using the mysql2 module. The connection configuration contains the host, user, password, and database name. When attempting to connect, the application reports either a success or an error message, ensuring that the database connection state is known. This connection is critical for storing and retrieving user information, as well as managing authentication.

Middleware binds the database connection to each request object, allowing route handlers to easily access the database. This approach ensures that all route handlers can conduct relevant database activities without requiring multiple connection setups.

The programme then defines and implements several route modules. IndexRouter and usersRouter handle generic routing, whereas additional routes such as registerRoutes, checkEmailRoute, dashboardRoute, booksRoute, profileRoute, exerciseRoute, and logoutRoute handle particular functionality. Each route module is initialised with a database connection to ensure smooth interaction with the database.

The programme supports user authentication via GET and POST routes for login. The GET route displays the login screen, whereas the POST route handles user login by accessing the database using the provided email and password. If the credentials are valid, the user session variables are set, and the user is sent to the dashboard. If not, an error message is shown.

The dashboard route confirms that the user is logged in by verifying the session. If the user is not authenticated, they will be sent to the login page. Otherwise, the dashboard view is displayed using the username as a variable to allow for personalised display.

a. Dashboard

```
doctype html
html
  head
    title Dashboard
    meta(charset="UTF-8")
    meta(name="viewport" content="width=device-width, initial-scale=1.0")
    link(rel="stylesheet" href="/stylesheets/dashboard.css")
  script(src="https://code.jquery.com/jquery-3.6.0.min.js")
  script.
    $(document).ready(function() {
      $('.dropdown').on('click', function() {
        $(this).toggleClass('open');
      });

      let inactivityTime = function () {
        let time;
        window.onload = resetTimer;
        window.onmousemove = resetTimer;
        window.onmousedown = resetTimer; // catches touchscreen presses as well
        window.ontouchstart = resetTimer; // catches touchscreen swipes as well
        window.ontouchmove = resetTimer; // required for touchscreen devices
        window.onclick = resetTimer;     // catches touchpad clicks as well
        window.onkeypress = resetTimer;

        function logout() {
          window.location.href = '/logout'; // redirects to logout route
        }

        function resetTimer() {
          clearTimeout(time);
          time = setTimeout(logout, 30000); // 30 seconds of inactivity
        }
      };

      inactivityTime();
    });
body
  .container
    header
      h1 Welcome, #{username}!
      nav
        ul.menu
          li
            a(href="/profile") Profile
          li.dropdown
            a(href="#") Menu
            ul.dropdown-menu
```

```
              li
                a(href="/exercises") Exercises
              li
                a(href="/books") Books
              li
                a(href="/logout") Logout

    main
      section
        h2 Introduction to Python
        p
          | Python is a powerful and versatile programming language that is easy to learn and fun to use.
          | Whether you are a beginner or an experienced programmer, Python has something to offer.
          | With Python, you can develop web applications, automate tasks, analyze data,
          | create machine learning models, and much more.
        p
          | Python's syntax is designed to be readable and straightforward,
          | which makes it an excellent choice for people who are new to programming.
          | The Python community is also very active and supportive, providing a wealth of resources
          | and libraries that you can use to enhance your projects.
      section
        h2 About Our Website
        p
          | Our website is dedicated to helping you learn and master Python programming.
          | We offer a variety of tutorials, exercises, and projects that are designed to teach you
          | the fundamentals of Python and help you apply what you have learned in real-world scenarios.
        p
          | You can track your progress through your profile, complete exercises to reinforce your learning,
          | and work on projects that will help you build a portfolio of Python applications.
          | Join our community of learners and start your journey to becoming a Python expert today!
```

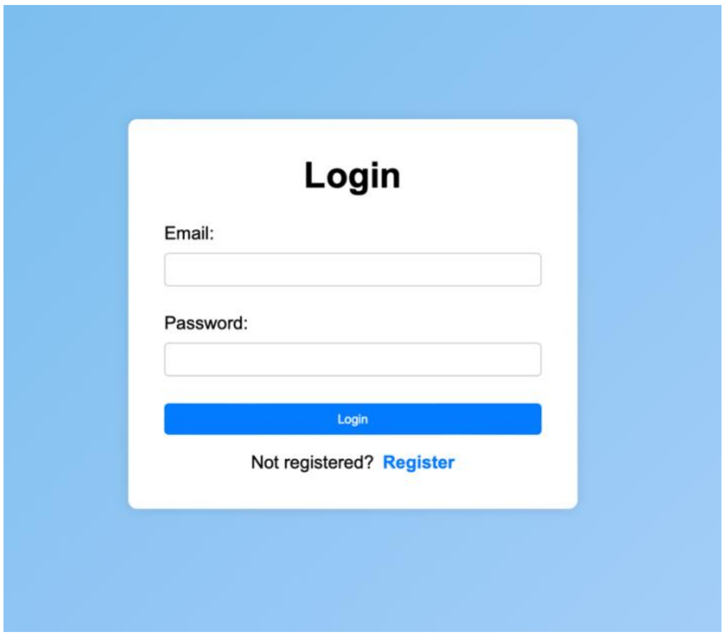
This Pug template is a dashboard page for a Python learning platform. The head section contains metadata for character encoding and viewport settings, as well as a link to a CSS stylesheet and the jQuery library. A script initialises a dropdown menu and activates an inactivity timer, logging the user out after 30 seconds of inactivity. The body includes a container with a header that displays a personalised welcome message, as well as a navigation menu with links to the profile, exercises, books, and a logout button. The primary content area includes parts that introduce Python and describe the goal, resources, and user advantages of the website.

NOTE: Any kind of breaches to a page before the user session starts through a log in will be redirected back to the login page

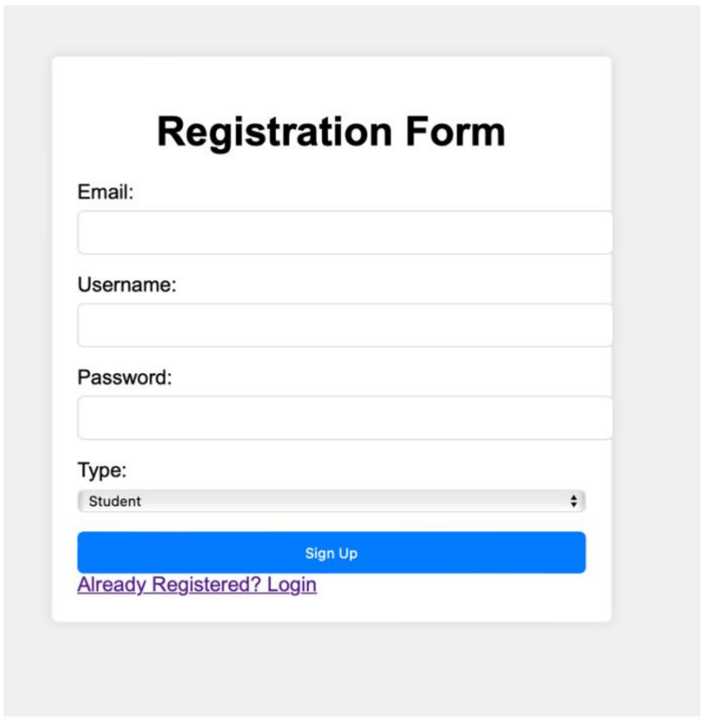
```
outer.get('/dashboard', (req, res) => {
   if (!req.session.user) {
     res.redirect('/login');
```

User Guidance

1. User Authentication



The user starts with a login page, if user is already registered they can enter the correct credentials to access the main page. If the user isn't registered yet they can click the register link to be redirected to the register page. If user enters the credentials wrongly they get a login failed message.


Registration Form

Email:

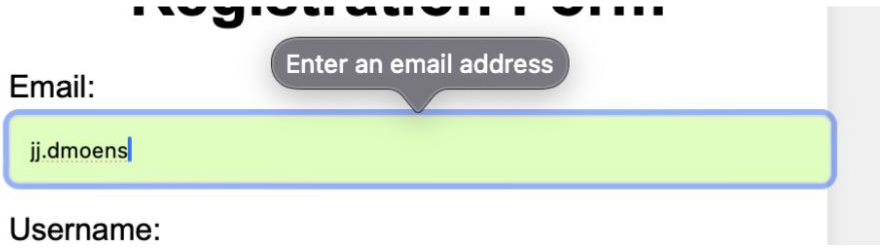
Username:

Password:

Type:

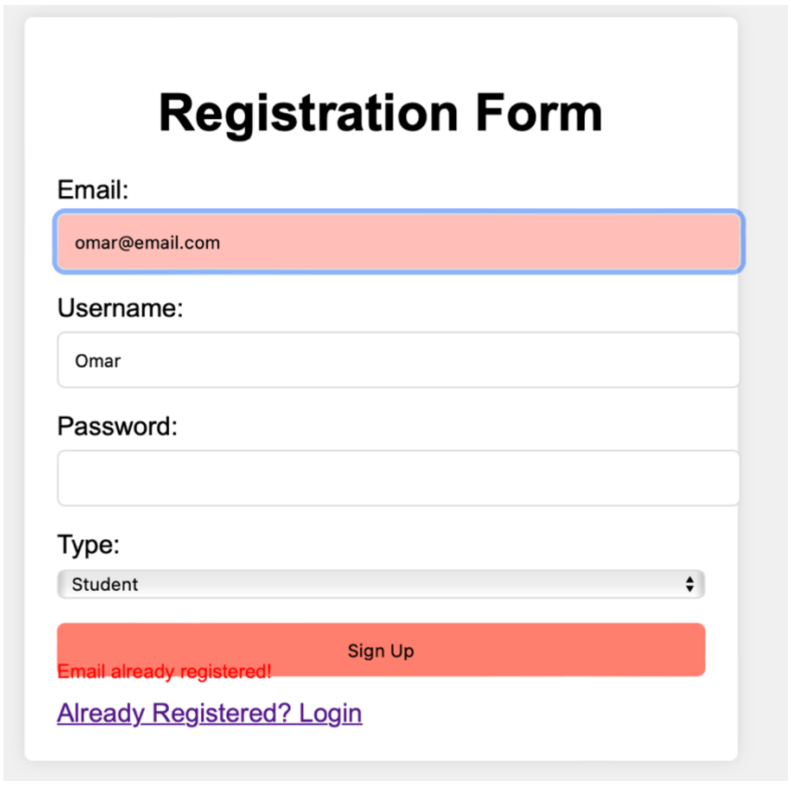
Student

In the registration page the user is prompted to enter his information. If the email format is entered wrongly the user is prompted to enter an email address, while if the email entered already exists the user will be prevented from registering and asked to enter an email address. If user is already registered they can click the login button to redirected to the login page.


Email: jj.dmoens

Enter an email address

Username:



Registration Form

Email:

omar@email.com

Username:

Omar

Password:

Type:

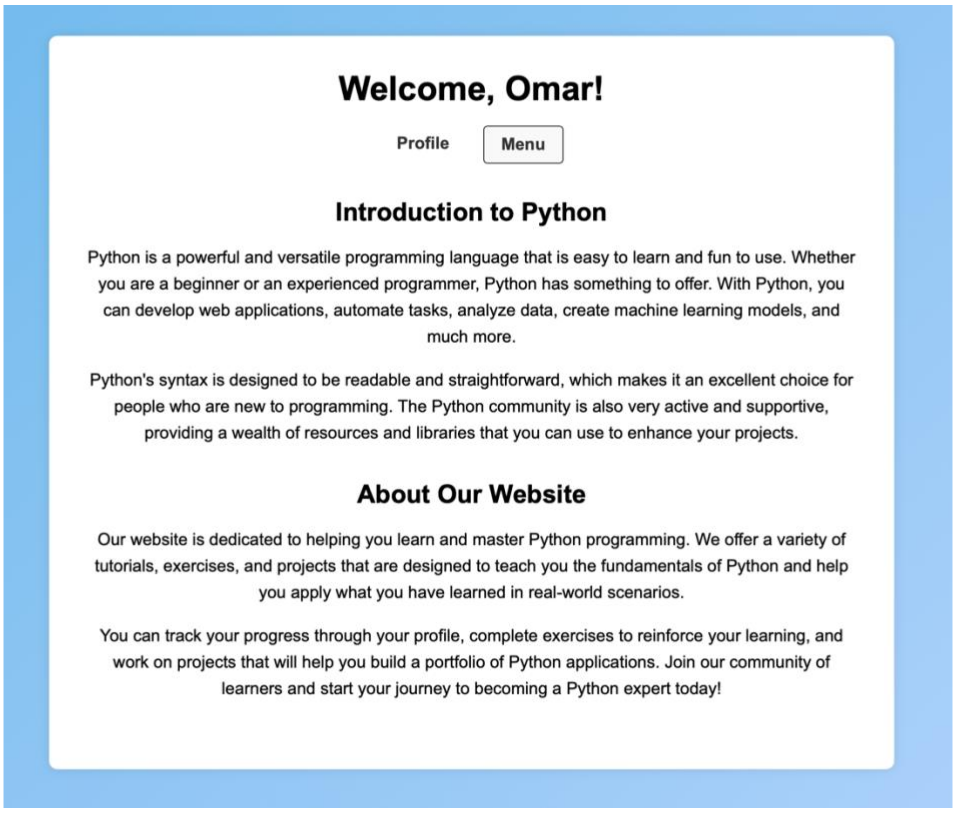
Student

Sign Up

Email already registered!

Already Registered? Login

2. Profile Management



Welcome, Omar!

Profile Menu

Introduction to Python

Python is a powerful and versatile programming language that is easy to learn and fun to use. Whether you are a beginner or an experienced programmer, Python has something to offer. With Python, you can develop web applications, automate tasks, analyze data, create machine learning models, and much more.

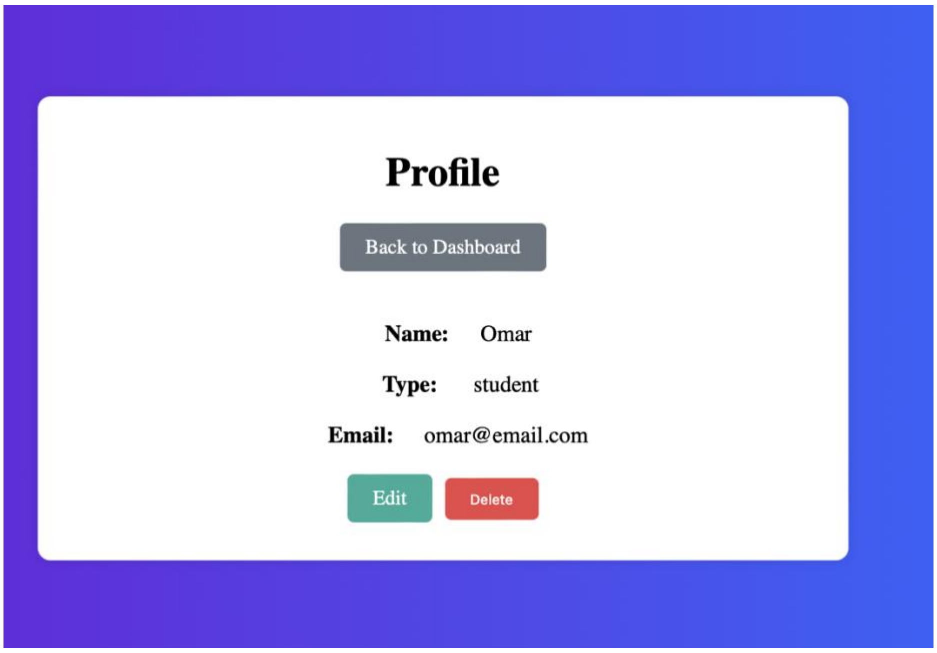
Python's syntax is designed to be readable and straightforward, which makes it an excellent choice for people who are new to programming. The Python community is also very active and supportive, providing a wealth of resources and libraries that you can use to enhance your projects.

About Our Website

Our website is dedicated to helping you learn and master Python programming. We offer a variety of tutorials, exercises, and projects that are designed to teach you the fundamentals of Python and help you apply what you have learned in real-world scenarios.

You can track your progress through your profile, complete exercises to reinforce your learning, and work on projects that will help you build a portfolio of Python applications. Join our community of learners and start your journey to becoming a Python expert today!

The main Page is the dashboard which redirects to most pages as well as having an introductory to python and the website.

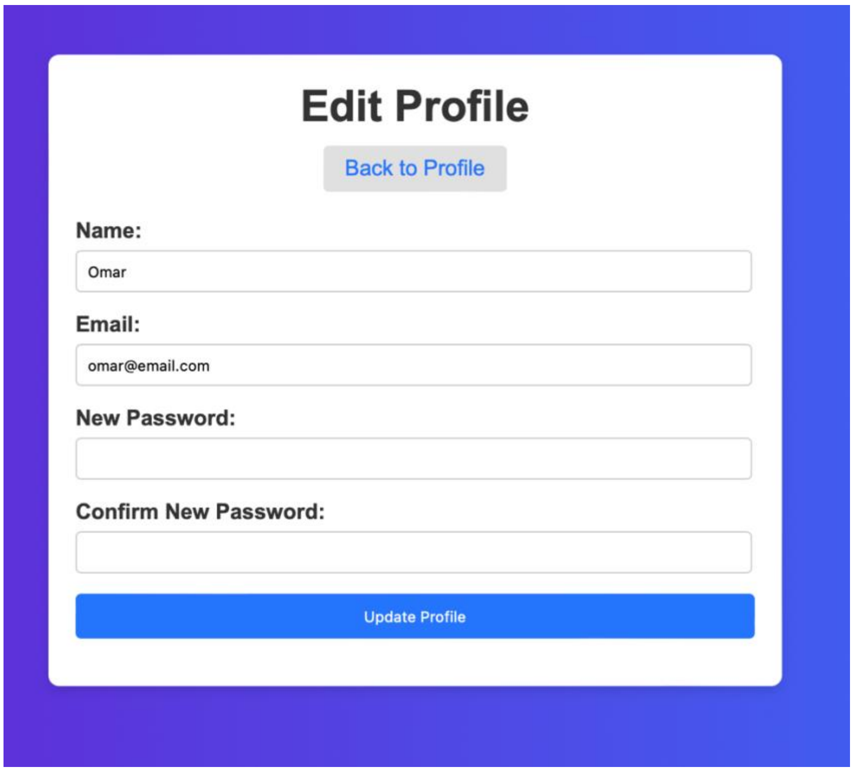


Name: Omar

Type: student

Email: omar@email.com

The profile page view the users information, the user can then go back to the dashboard, click the edit button which redirects them to an edit credentials page or delete button which gives a pop up deletion confirmation that if user clicks yes the profile is deleted and credentials are erased from the db.



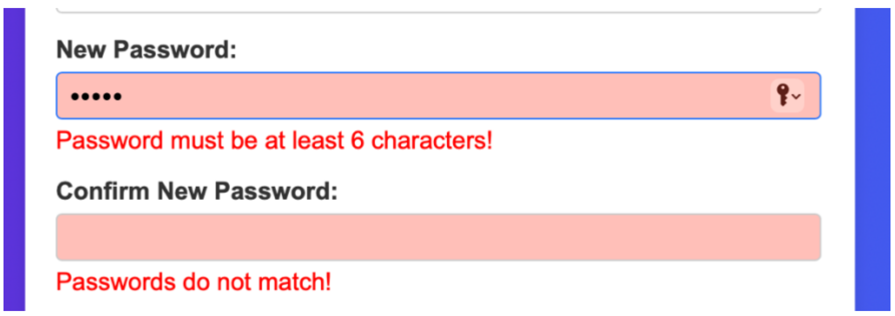
Name: Omar

Email: omar@email.com

New Password:

Confirm New Password:

When in the edit page the user can edit name, email or password.



New Password:

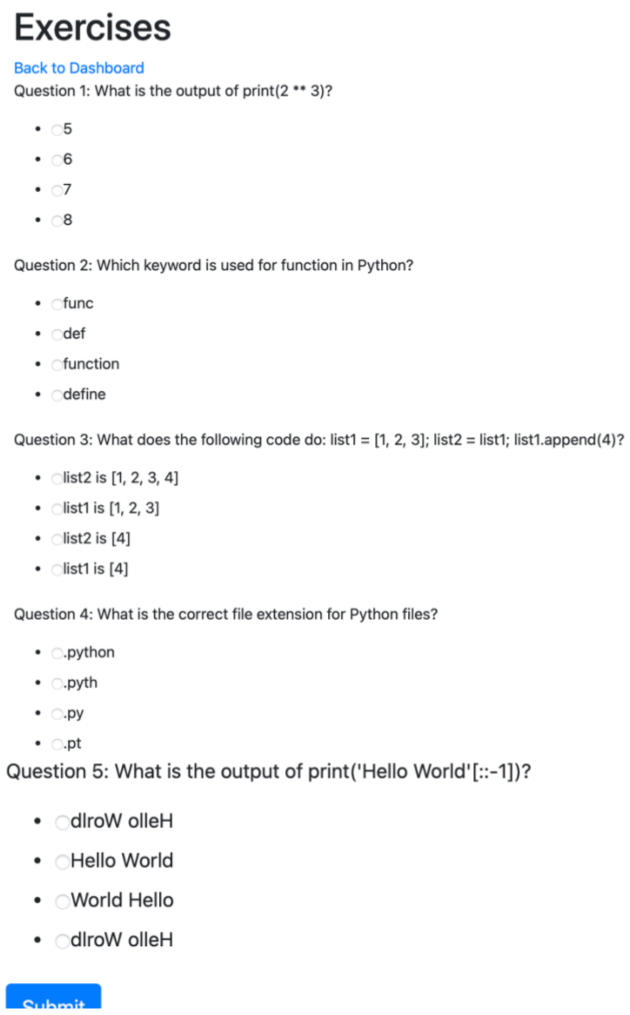
•••••

Password must be at least 6 characters!

Confirm New Password:

Passwords do not match!

When changing the password the user needs to enter at least 6 characters and rewriting the same password for confirmation.



Exercises

Back to Dashboard

Question 1: What is the output of print(2 ** 3)?

- 5
- 6
- 7
- 8

Question 2: Which keyword is used for function in Python?

- func
- def
- function
- define

Question 3: What does the following code do: list1 = [1, 2, 3]; list2 = list1; list1.append(4)?

- list2 is [1, 2, 3, 4]
- list1 is [1, 2, 3]
- list2 is [4]
- list1 is [4]

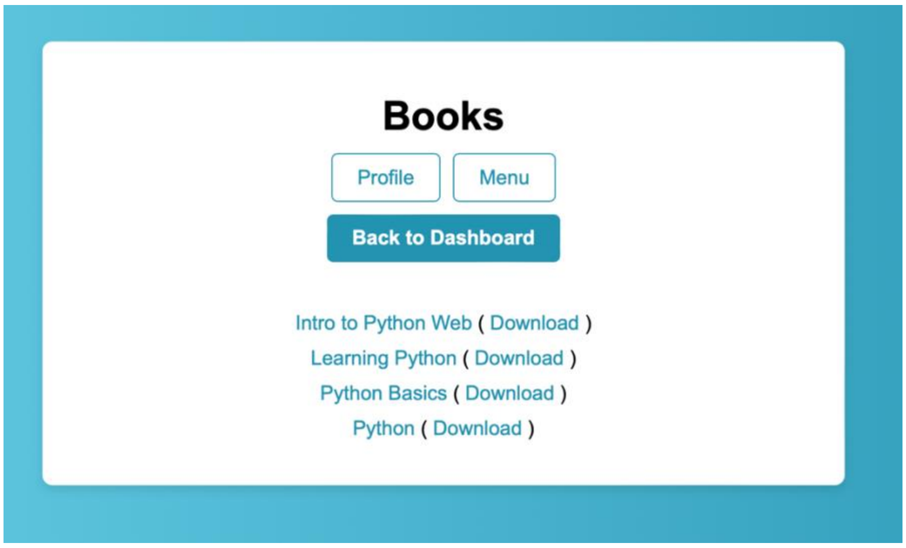
Question 4: What is the correct file extension for Python files?

- .python
- .pyth
- .py
- .pt

Question 5: What is the output of print('Hello World'[::-1])?

- dlroW olleH
- Hello World
- World Hello
- dlroW olleH

When clicking exercise from the menu in dashboard user is redirected to the exercise page to do a quick quiz and submitting the quiz for results.


Books
Profile
Menu
Back to Dashboard
Intro to Python Web (Download)
Learning Python (Download)
Python Basics (Download)
Python (Download)

The user can also access books through the book link to view or download Python pdfs.

Conclusion

Pyledge is a comprehensive and user-friendly educational environment for Python programming. Pyledge's solid design and well-thought-out features provide a balanced and successful learning environment for both students and instructors.

The platform includes a variety of features such as user identification, profile management, access to a library of Python books, and interactive exercises to encourage learning. The usage of Express.js for server-side functionality, along with MySQL for data storage, results in a smooth and efficient user experience. The use of Pug templates for dynamic content rendering improves the visual appeal and usability of the site.

In conclusion, Pyledge stands out as a well-rounded instructional tool that not only makes Python learning easy and pleasant, but also assists in providing quality instruction. Pyledge's infrastructure, along with the features, establishes it as a platform for anyone seeking to master Python programming.

References

GitHub. (n.d.). Retrieved from GitHub: https://github.com
draw.io. (n.d.). *draw.io*. Retrieved from drawio: https://www.drawio.com
draw io. (n.d.). Retrieved from diagrams: https://app.diagrams.net/?src=about
figma. (n.d.). Retrieved from figma: https://www.figma.com
OpenJs Foundation. (n.d.). Retrieved from Node.Js: https://nodejs.org/en/download/package-manager
Pug. (n.d.). Retrieved from pug: https://pugjs.org/api/getting-started.html