Project Rules

General Description

- This course and tutorials will guide you through the development of a simple e-commerce application. Content- and effort-wise, we split this project into five Parts.
- The project involves development of specific functionalities of a web application. The description of each part of the project will contain a small back story to provide some context for the functionality.
- Although we suggest a timeline for the implementation of each part, it is up to you when to implement them.
- Each Part builds on the previous one. Thus, if you want to implement Part 3, you will need the content corresponding to Part 1 and Part 2 as well.
- You can use one of the two following frameworks: Python/Django or Node/Express. No other languages or frameworks are allowed.
- Your project must be hosted at our Gitlab server (URL: `https://gitlab.swd.appsec.saarland`).
- Each team must create one Gitlab repository. The name of the repository must correspond to the name of the team specified during team registrations, unless the name contains non-printable characters that are rejected/removed by Gitlab or is too complex.
- Teams can use the Gitlab repository as they prefer to structure and track the development effort, e.g., by using branches and issues.

Deadlines and Submission Rules

- The five parts must be handed in in two different moments using the CMS (hard deadlines):
  1. The first deadline is on Monday, 31st of August 2020, 00:00AM. By this date, you need to hand in Part 1 and Part 2.
  2. The second and last deadline is on Friday, 4th of September 2020, 00:00AM. By this date, you need to hand in Part 3, Part 4, Part 5.
- For each deadline, your team will upload a text (.txt) file to the CMS, containing a single GitLab URL for the exact repository version. The submitted URL will correspond to the exact Git commit that you want to be evaluated.
- To find the URL to submit:
  1. Go to your project GitLab repository web page.
  2. Click on the `Repository->Commits` option on the left-hand side menu. This page will list all Git commits your team performed when working on the assignments. If you are using a branch other than master, choose your branch from the select button on the top (the default branch is master).
  3. Click on the commit that implements the parts required by the submission and that should be considered for evaluation. The new page will show the list of changes after the selected commit.
  4. Click on the button "Browse files"", on the top right. The new page will show the snapshot of the repository after the selected commit.

5. Copy the URL in the browser's navigation bar into the text file, and upload the file in the CMS. The URL would be similar to `https://gitlab.swd.appsec.saarland/testuser/testproject/-/tree/c3c162fbaf3a`, where `c3c162fbaf3a` is the unique SHA value of the commit.
6. Please, double check that the repository snapshot accessible via the URL correspond to the source code you want us to evaluate.

- Previous or later commits will not count towards that deadline.
- Your team should appoint a single member to do the submission of each deadline; this person should be the same for all submissions.

General Requirements

- When evaluating submissions, we will access each web application using this URL: `http://127.0.0.1:3000`. Before each submission, make sure that your web application is serving responses to that URL. This is a MUST-HAVE requirement.
- For each project part, we will indicate additional endpoints where the various services MUST be accessible (e.g., the login service, or how to fetch the products of your shop). Such naming conventions are binding and MUST be respected. Similarly, we will outline database data models (e.g., User or Product). Such specifications are also MUST-HAVE features. We will highlight every MUST requirement using the symbol ◄.
- For the sake of clarity, names as specified in the user stories are not binding (unless we specifically use the same names later in the Task descriptions).

Part 1

The goal of the first part is to implement the login and registration features.

Introduction

You have just been hired by a company developing web applications for other businesses. Recently, a new client has contacted your employer and asked for a new e-commerce application where they can sell some of their futuristic products. Right now, the client has been very vague about the web app specifications... but what is for sure is that they do not want anonymous users purchasing products on their website, but rather, they prefer keeping records of shopping history and of each user. Since you are the newbie and that there is not so much time pressure, they ask you to start working on this project by implementing the user management part of the e-commerce application.

Your older colleagues already did some modeling work for you, and these are the specifications that they give you. The e-commerce web app (let us call it *thin air LTD*) needs to have a home page or landing page reachable at `https://thin-air.appsec.saarland /accounts`. From here, the user can either log in, or register; if they already have an active session, we can momentarily show a welcome message followed by their username.

- By clicking on the login button, the user is redirected to another URL (`https://thin-air.appsec.saarland /accounts/login`) where they can log in. They will enter their username, password and confirm with a submit button.
- By clicking on the registration button, the user is again redirected to a third URL (`https://thin-air.appsec.saarland /accounts/registration`). Here, they just provide their email address (used as username), first and last name, and password.

As this is not much work, they asked you to provide a simple user interface that you can show to the client during the next meeting, to show a couple of features and ask for feedback. Of course, this means that the website has to be fully functional, and you need to be able to register actual users (even if dummy ones) and show the client that data is stored in the back-end database. Moreover, you would like to leave a good impression on your colleagues, and implement these services by paying attention to the security features (e.g., by doing a two-step registration phase).

Note. As we are still in the development phase of our ecommerce web application, we expect that all specified URLs of our site `https://thin-air.appsec.saarland` be available at `http://127.0.0.1:3000`. For simplicity, hereafter we specify the URLs *relative* to the site origin.

Task 1.1 Setup Database and ORM

For the purpose of this course, we will use a simple `SQLite` database:

Django Framework If you are using Django, `SQLite` is the default database and you can use the Django built-in ORM to connect to and query your database. For further information, see `https://docs.djangoproject.com/en/3.0/intro/tutorial02/`.

Express Framework If you are using NodeJS and Express, you should use `TypeORM`, a powerful ORM for NodeJS applications. Further documentation is available at `https://typeorm.io`. You can install it easily with a few `npm` commands:

- **Installation:**
  1. `npm install typeorm --save`

2. `npm install reflect-metadata --save`
3. `npm install @types/node --save`
4. `npm install sqlite3 --save`

- **Usage:** to create database entities using `TypeORM` in **JavaScript**, You need to create a *entity* folder containing all your database models with a `.js` file extension (e.g., `User.js`). For quick setup, see this example repository: `https://github.com/typeorm/javascript-example/tree/master/src/app1-es5`.

Task 1.2 Setup your Project Structure

Please note the following naming conventions:

- Name of the web application: simple_ecommerce ◄
- Name of the module for user management: user_mgmt ◄

Django Framework If you are using Django, start your project with the following commands:

1. `django-admin startproject simple_ecommerce`
2. `django-admin startapp user_mgmt`

Note that we create a user management app (i.e., the `user_mgmt` folder) which should contain the logic for user login and registration.

Express Framework If you are using NodeJS and Express, you can start your project with Express application generator:

1. `npm install -g express-generator`
2. `express --view=pug simple_ecommerce`

Then, simply create a `user_mgmt.js` file inside the routes folder which should contain the logic for user login and registration. For further help, see `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/skeleton_website`.

Hint. You can consider installing the `nodemon` package to automatically restart your NodeJS server upon changes in your code. To do so:

1. Run: `npm install --save-dev nodemon`
2. Add the below key-value entry to the `scripts` section of your `package.json` file:
   - `"devstart": "NODE_ENV=development node_modules/.bin/nodemon ./bin/www"`.
3. Run your application with: `npm run devstart`.

Task 1.3 Create a User Model

Create a user database model with the following fields:

- username*: email address of the user
- password*
- first_name*
- last_name*
- datetime_joined+: the date on which the user joined your web site.
- enabled*: a boolean specifying if the account's email address is verified.
- activation_token+

Note. Fields marked with asterisks (*) must be entered by the user from the user interface. Fields marked with a plus symbol (+) must be auto-generated by the backend. Please consider using the above-specified names as a binding naming convention (◄).

Task 1.4 Web Pages and URL Formats

In this assignment, you will create three web pages:

1. A login page accessible at: `/accounts/login`◄.
2. A registration page accessible at: `/accounts/registration`◄.
3. A home page accessible at: `/index`◄.
    - If the user is logged in, the home page will contain a logout link: `/accounts/logout`◄.

Note 1. For each of the previous web pages of Task 1.4, you need to implement both a user interface (through an HTML page) and a routine handling the functionality. Data must be persisted to the database.

Note 2. Logged users should be redirected to the index page with their names shown on top of the page.

Note 3. The created accounts should be disabled after initial registration (i.e., `enabled: false`). You must verify registered accounts through `/accounts/<email>/verify/<token>`◄, the concrete value of which is sent, in reality, to the user's claimed mail address upon registration. For the purpose of this assignment, you should just print the activation link in the standard output.

Note 4. Think about how you store the password in the database and how you generate the activation token.