

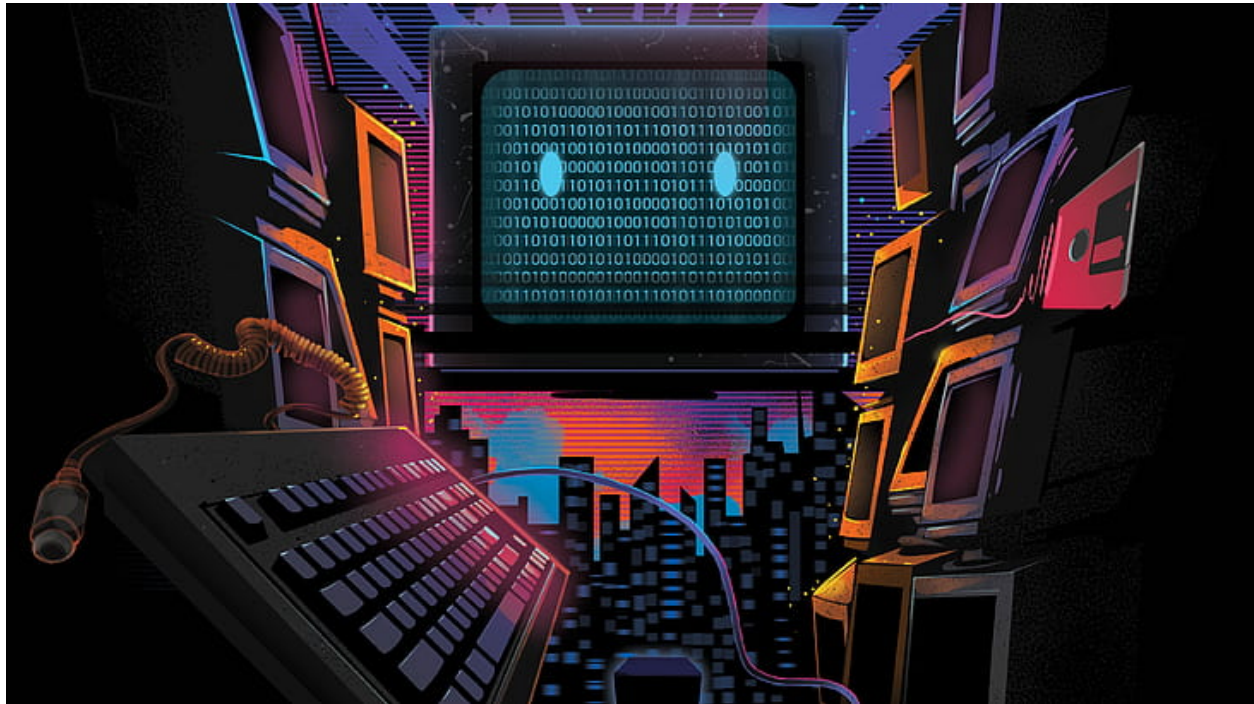
# Software design

## Code-Next

### System Design Document

#### CSC301 - Introduction to software engineering

Collaborators: Abdullah, Sana, Kyle, Amelia, Melissa, Mohammad, Tamseel



# Table of Contents

---

<u>Content</u>	<u>Pages</u>
Front-end CRC models	3
Back-end CRC models	7
Software Architecture	12

## Front-end CRC models

---

### SignInPage

React Component(s): SignInPage	
Parent class (if any): React.Component	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"><li>→ To provide a user interface for users to sign up for the website.</li><li>→ To collect user information such as username, and password.</li><li>→ To validate user information to ensure they are entered correctly.</li><li>→ To send the collected information to the server for processing.</li><li>→ To redirect the user to the home page once signed in.</li></ul>	RegisterPage

### RegisterPage

React Component(s): RegisterPage	
Parent class (if any): React.Component	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"><li>→ To provide a form for users to enter their personal information, such as username, email, and password</li><li>→ To validate user information to ensure they are entered correctly.</li><li>→ To send the collected information to the server for registration.</li><li>→ To receive and handle the registration response from the server</li><li>→ To redirect the user to the home page once signed in.</li></ul>	SignInPage

## NavBar

React Component(s): NavBar	
Parent class (if any): React.Component	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ To display navigation links</li> <li>→ To handle user interactions with navigation links (e.g. redirects to specific page)</li> </ul>	

## Dashboard questions and the individual Leet-code question

React Component(s): Dashboard, LeetQuestion	
Parent class (if any): React.Component	
Subclasses (if any): LeetQuestion	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ The component receives props from its parents component (Dashboard) and renders the data appropriately.</li> <li>→ Uses a container to store all the questions displayed for the user, and renders the questions in the UI</li> <li>→ Uses the CodeNow button to trigger the navigation to the next page when clicked.</li> </ul>	Questions

## Leetcode questionnaire form

React Component(s): Questions	
Parent class (if any): React.Component	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ To display a single question to the user and track whether the question has been selected to start or not.</li> <li>→ To ask the user if they have completed the chosen Leetcode question.</li> <li>→ To record the perceived difficulty level by the user.</li> <li>→ To record the time taken by the user to attempt/complete the question.</li> </ul>	

## About

React Component(s): About	
Parent class (if any): React.Component	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ To display information about spaced repetition.</li> <li>→ To provide users with an understanding of the benefits of spaced repetition.</li> <li>→ To provide users with information about how the platform will utilize spaced repetition to avoid the forgetting curve.</li> <li>→ To present the information in a visually appealing and organized way.</li> </ul>	<p>NavBar</p>

## Pre-quiz

React Component(s): PreQuiz	
Parent class (if any): React.Component Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"><li>→ To analyze users strengths and weaknesses.</li><li>→ To generate questions based on the users skill level.</li><li>→ To help users get started with their LeetCode journey.</li></ul>	<div>Dashboard</div> <div>LeetQuestion</div>

## Notifications

React Component(s): Notifications	
Parent class (if any): React.Component Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"><li>→ To display a notification to remind the user to repeat a question they have done or attempted before.</li></ul>	

## Back-end CRC models

---

React Component(s): Attempts	
Parent class (if any): models.Model	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"><li>→ This class represents the database model that will store the user-submitted feedback after a problem is attempted.</li><li>→ To track which problems have been attempted</li><li>→ To track when those attempts occurred and to store a brief survey on the user's thoughts of the question.</li></ul>	

React Component(s): AttemptView	
Parent class (if any): APIView	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"><li>→ This class handles the http get and post endpoint /problems/attempt and will create a new attempt record in the database when posted to.</li><li>→ When a get request is received, if the problem_id parameter is provided, then it will return all of a user's attempts at that particular problem.</li><li>→ If a problem ID is not provided, all of the user's attempts will be returned.</li></ul>	Attempts class for the database model.

React Component(s): Categories	
Parent class (if any): models.Model	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ This class represents the database model that will store the categories that questions on CodeNow will be grouped into.</li> <li>→ Instead of categorizing questions on our own, Codenow like Coursera, will survey users after completing a question to determine which category a question should be in.</li> </ul>	

React Component(s): CategoryView	
Parent class (if any): APIView	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ This class handles the http get request for /problem/categories and will return a Json object containing a list of all problem category names, descriptions and ids.</li> </ul>	Categories class for the database model



React Component(s): Problem	
Parent class (if any): models.Model	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ This class represents the database model that will store problems.</li> <li>→ The problems themselves are hosted on leetcode, this database model stores a link to that problem along with necessary problem metadata such as the problem name, category and id.</li> </ul>	

React Component(s): ProblemsView	
Parent class (if any): APIView	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ This class handles HTTP get and post requests for our problem endpoint.</li> <li>→ Get requests will retrieve a list of all available questions and relevant metadata such as name, description and category.</li> <li>→ A post request to this endpoint will add a new question to our database.</li> <li>→ All requests and responses will be done in JSON.</li> </ul>	Problem class for the database model.

React Component(s): LoginView	
Parent class (if any): APIView Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ This class handles the authentication token when logging into an account through the HTTP endpoint login.</li> <li>→ The data is grabbed from the fields in the serializer and a token is generated through the information taken.</li> <li>→ Once the token is generated, it is returned as an HTTP response.</li> </ul>	

React Component(s): PrequizProblemsView	
Parent class (if any): APIView Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ This class handles GET'ing the required prequiz problems for the prequiz and POST'ing the data once the form is filled on the frontend.</li> <li>→ Three random problems are grabbed from the database on GET</li> <li>→ Their properties perceived_difficulty and completion time are updated on POST.</li> </ul>	

React Component(s): AddProblemView	
Parent class (if any): APIView	
Subclasses (if any): N/A	
Responsibilities	Collaborators
<ul style="list-style-type: none"> <li>→ This class handles only adding problems to the database.</li> <li>→ Populates the rows with their question id, problem name, difficulty and url.</li> </ul>	

# Software Architecture Diagram

This project adheres to a Three Tier architecture with React, Django, and SQLite.

The architecture diagram for using React, Django, and SQLite for a web application typically looks something like this:

1. Front-end: React is used to build the user interface of the web application. It is a JavaScript library for building user interfaces, and it will allow us for building reusable UI components and managing the state of the application.
2. Back-end: Django is used as the back-end framework for the web application. It provides an easy way to handle HTTP requests and responses, manage the database, and implement the business logic of the application.
3. Database: SQLite is used as the database management system for the web application. It is a serverless, file-based database that provides reliable and efficient data storage and retrieval capabilities.

These three technologies, when combined, offer a powerful and efficient solution for developing this website, making it a great choice for the project.

References (linked below):

- [Reference 1](#)
- [Reference 2](#)

