GENERAL SIR JOHN KOTELAWALA DEFENCE UNIVERSITY (KDU) DEPARTMENT OF INFORMATION TECHNOLOGY

IT2053 RAPID APPLICATION DEVELOPMENT

Assignment Part 5 -Food Recipe System Report

ASSIGNMENT PART 5
GROUP NO: 29 STUDENT ID:
D/BIT/24/0024 - A.KESHANI RANGI SILVA
D/BIT/24/0057 - S H HETTIARACHCHI
D/BIS/24/0003 - G.Y.S.WICKRAMARATNE
D/BIS/24/0039 - K.L.C.JEEWANANDA

Project Description

A web application named Food Recipe System discovers recipes and saves them as well as share their recipes with others. Serves migrant students who struggle with meal preparation because of their limited cooking skills during this. Through its interface, users can browse recipes through categories to explore detailed listings of recipes which can be saved for later use.

The Food Recipe System provides access to several elements.

1.User Authentication: Secure login and registration functionality

2. Discovery: Browse recipes by categories (Pasta, Rice, Curry, etc.)

3. Users can easily access recipe details about ingredients together with preparation steps and timings and levels of difficulty.

4. Users experience engaging with the platform through rating recipes and adding comments.

5.Personal Collections: Save favorite recipes to a personal collection

6.The search function enables users to locate recipes keywords and ingredients and categories.

The application implements device compatibility enabling use through desktops mobile phones and tablets.

MVC Implementation Details

The Food Recipe System adopts Model-View-Controller (MVC) as its architectural structure to split application logic into three modular elements thus enabling better code organization with reuse potential and improved maintainability.

Model (Data Layer)

The Model component manages all data operations and handles the business rules.

The system manages user information through its authentication credentials and stored recipes which are stored in the User Model component.

The Recipe Model acts as a data representation which includes properties for title, description and ingredients as well as instructions, cooking details and metadata.

The application stores user-generated comments regarding recipe information through its Comment Model component.

The Datastore Class functions as a database simulator to manage data persistence functions and retrieval processes.

Since it lacks a conventional database, the application deploys an in-memory data store which runs through Python code. Through its Datastore class the system offers multiple methods to serve the following functions:

- User registration and authentication
- Recipe retrieval, filtering, and searching
- Comment management
- Saved recipe tracking python

```
# Example of Model code structure

class Recipe:
    def __init__(self, id, title, description, image, prep_time, cook_time,
                 servings, difficulty, ingredients, instructions,
                 categories, author, date, ratings=None, featured=False):
        # Property initialization
        self.id = id
        self.title = title
        # Other properties...
        self.comments = []
```

View (Presentation Layer)

The View component handles the user interface using Flask's Jinja2 template engine:

- **Base Templates**: Provide the common layout and structure
  - base.html: Main template for content pages
  - base_auth.html: Specialized template for authentication pages
- **Content Templates**: Extend the base templates to display specific content
  - index.html: Homepage with featured recipes and categories
  - recipe.html: Detailed recipe view
  - categories.html: Category-based recipe browsing
  - saved-recipes.html: User's saved recipe collection
- **Authentication Templates**: User login and registration

- login.html: User login form
- register.html: New user registration form

Templates use HTML5, CSS3, and JavaScript to create a responsive and interactive user interface. The styling follows a consistent design language with a focus on usability and aesthetics.

html

```
<!-- Example of View template structure -->
{% extends "base.html" %}
{% block content %}
  <section class="hero">
    <div class="container">
      <div class="hero-content">
        <h2>Let's Eat Quality Food</h2>
        <p>Discover thousands of recipes from around the world</p>
      </div>
    </div>
  </section>
  <!-- More content... -->
{% endblock %}
```

Controller (Logic Layer)

The Controller component manages the application flow, handling HTTP requests and coordinating between the Model and View:

- **Route Handlers**: Process user requests and determine appropriate responses
- **Form Processing**: Handle user input from forms and validate data
- **Session Management**: Track user authentication state
- **API Endpoints**: Provide data for AJAX calls to enhance interactivity

The Controller is implemented using Flask routes in app.py, which direct requests to the appropriate functions and render the correct templates with the necessary data.

```
python

# Example of Controller code structure

@app.route('/recipe/<int:recipe_id>')

def recipe_detail(recipe_id):

    recipe = data_store.get_recipe_by_id(recipe_id)

    if not recipe:

        return render_template('404.html'), 404

    return render_template('recipe.html', recipe=recipe)
```

Challenges Encountered and Solutions

During development, some challenges were encountered and addressed:

1. Simulation of Database Functionality

Challenge: Designing the system on a database-independent platform, data persistence was therefore a challenge.

Solution: Utilized an in-memory data store based on Python dictionaries to simulate database functionality. Implemented Datastore class methods for CRUD operations to facilitate database-like data interaction with data in memory during application run-time.

2. Template Integration and Path Handling

Challenge: File path management and proper template integration were challenging, particularly for static files and template inheritance.

Solution: Applied the standard approach to handling paths with Flask's url_for() function for all internal URLs and static files. Applied a base template framework with correct inheritance to ensure consistency across pages without code duplication.

3. User Authentication Without a Database

Challenge: User authentication without a persistent database was difficult.

Solution: Used Flask sessions to manage the user login state and the in-memory store to save user credentials. While not suitable for production, it sufficed for the demo project.

4. Responsive Design Implementation

Challenge: Ensuring a consistent user interface across different device screen sizes was the top priority.

Solution: Used a mobile-first design approach using CSS media queries to ensure the app performed optimally on all devices. Utilized a grid-based CSS layout system that resized dynamically based on screen size and provided easy access to content as well as appearance on any device.

5. HTML Form Integration with Flask

Problem: Integrating HTML forms with Flask routes and properly processing the data in the right way was a little tricky, mainly maintaining state and handling error conditions.

Solution: Used standardized form submission via POST methods and implemented proper error handling and validation. Utilized Flask's request object to handle form data and provided user feedback via flash messages and template variables.

Conclusion -The Food Recipe System implements the MVC pattern to develop an up-and-running web application for recipe management. With the concerns separated into different entities, the project achieves a clean and sustainable codebase which breezes to include future features.

The problems that occurred during development were a rich learning experience in using Flask, applying MVC techniques, and designing web applications in a responsive mode. Practices implemented indicate attention to good practice in web development alongside the capacity to overcome technical challenges through creative problem-solving.