ITCS 4155

5GB: Design Document

Purpose Statement

The purpose of this Software Design Document (SDD) is to provide a comprehensive overview of the "Find My Hobby" web application. It serves to clearly communicate the architecture, components, design decisions, and functionality of the system to developers, stakeholders, and future maintainers. This document enables consistent implementation without requiring major design decisions and acts as a reference for understanding, evaluating, and enhancing the application over time.

1. Project Overview

"Find My Hobby" is a web application designed to help users discover new hobbies based on their interests. The application targets users who are seeking engaging and fulfilling pastimes but are unsure where to start. Stakeholders include hobby enthusiasts, beginners looking for a new activity, and administrators maintaining the hobby database. By guiding users through a short personality quiz and matching them with hobby categories and events, the application makes hobby discovery accessible and interactive...

Team Members

- Scrum Master: Natalie Tepedino
- Product Owner: Yadhira Marcos-Avila
- Developers: Adam Kerns, Leo Amromine, Michael Gohn, Manny Campbell, Sasank Pagadala

2. Architectural Overview

The architecture follows a standard three-tier structure: presentation, logic, and data layers.

Alternatives Considered

Monolithic designs were rejected due to their poor modularity. The MVC (Model-View-Controller) pattern was adopted to ensure clean separation of concerns and improved maintainability.

2.1 Subsystem Architecture

Application-Specific Layer

- app.py: Routes and controller logic with Flask

- HTML pages (e.g., homepage.html, login.html, signup.html, hobby-quiz.html, Results-Events.html): Views rendered via Jinja2
- questions.js: Loads quiz questions dynamically

Application-Generic Layer

- models.py: SQLAlchemy models for User, Questions, Results, Events

Middleware Layer

- Flask (with Jinja2 templating and Flask-SQLAlchemy)

System Software Layer

- SQLite (defined via database_schema.sql)

Architectural Style: Model-View-Controller (MVC)

- **Model:** SQLAlchemy-based classes in models.py
- **View:** HTML/CSS with JavaScript and Jinja templates
- **Controller:** Flask-based routes in app.py

2.2 Deployment Architecture

This software will run on a single processor.

2.3 Persistent Data Storage

Relational database (SQLite) as defined in database_schema.sql:

- users: email, password
- questions: question text and options
- results: user hobby match
- events: hobby-related events and descriptions

2.4 Global Control Flow

- **Event-driven:** Interaction is driven by user events (e.g., button clicks)
- **No time dependency**: App is responsive, but not real-time
- **Single-threaded execution**: No concurrent threads used

3. Detailed System Design

3.1 Static View

Core Classes from models.py

User

- Attributes: id, email, password
- Methods: register(), login()

Question

- Attributes: id, question_text, options
- Methods: get_questions()

Result

- Attributes: id, user_id, matched_hobby
- Methods: store_result(), get_user_result()

Event

- Attributes: id, hobby, title, description, time, location
- Methods: get_events_by_hobby()

UI Structure (from HTML)

- homepage.html: Welcome page + How it works
- hobby.html: Hobby categories + community posts
- hobby-quiz.html: Quiz interaction
- Results-Events.html: Dynamic results and events
- login.html / signup.html: Auth pages
- styles.css: Full design language and responsive layout

User Stories (from UserStories.txt)

- Dark mode for accessibility
- Icon-based intuitive design for usability

3.2 Dynamic View

Quiz to Result Flow

1. questions.js dynamically loads questions.
2. User interacts via hobby-quiz.html.
3. app.py receives submitted answers via POST.
4. Server determines best-matched hobby.
5. Results-Events.html displays result with dynamic events.

Build Instructions

(from README.md)

- Download files into appropriate folders
- Open homepage.html to launch locally

Requirements

(from requirements.txt)

- Flask 3.1.0
- Flask-SQLAlchemy 3.1.1
- Jinja2, Werkzeug, SQLAlchemy, and more are listed in the file

Generative AI model used:

OpenAI. (2025). *ChatGPT (April 23 version)* [Large language model]. https://chat.openai.com/