# Project Documentation
# Engineering 4 Group 2

## 1. Introduction

### 1.1. Aims and objectives of the project

As a group, develop and deliver within a given timeframe an application integrated with an external API that solves a problem.

The project aims to create an app that helps users manage household tasks and set personal goals for hobbies or routines. The app allows users to assign tasks to themselves or others and even enjoy music while doing them, with playlists tailored to different types of tasks. Key features of the app include the ability for multiple users to share the app, view and manage both personal and joint tasks, and set and update personal goals. The app integrates with Spotipy (the Spotify API for Python) to find and play music that suits each task.

### 1.2. Roadmap of the Report:

- **Introduction**: project aims and objectives
- **Background**: an overview of the project's context and relevance.
- **Specifications and Design**: a detailed look at the system's requirements, design, and architecture.
- **Implementation and Execution**: insights into the development process, tools used, and challenges faced.
- **Testing and Evaluation**: summary of the testing strategies applied and the system's limitations.
- **Conclusion**: final thoughts on the project.

## 2. Background

“The Manager” app is designed to help users organize and track their household responsibilities and personal goals by adding accountability and motivation. Users are able to create, claim and update tasks, set and update goals, view all permitted records and listen to Spotify playlists preselected for each activity.

The app not only simplifies household chores by involving all family members but also helps reduce conflicts and ensures tasks are completed and not forgotten. It keeps users organized during busy weeks, helping them find time to focus on their hobbies or work

toward larger goals. Beyond these benefits, the app promotes better time management by allowing users to balance responsibilities with personal growth activities. It also fosters collaboration within households, encouraging teamwork and shared responsibility. Furthermore, the integration of music can make tasks more enjoyable, turning mundane chores into a more pleasant experience.

# 3. Specifications and Design

## 3.1. Functional Requirements:

- User can sign up, log in and log out
- User can add, update and view tasks and goals
- User can archive tasks
- User can claim tasks
- User can listen to Spotify playlists
- Application needs to fetch, store, and update data in the database

## 3.2. Non-Functional Requirements:

- The application must be secure with session protection
- The application must be user-friendly with a responsive design
- The application should have a scalable architecture

## 3.3. Design and architecture

- **Design**
    - **Manager.sql**: database – data storage and management
    - **Managertest.sql**: testing database
    - **Tests**: login, app and Spotify API tests
    - **Congif.py**: Spotify configuration
    - **Models.py**: database models, data management through SQLAlchemy
    - **Forms.py**: Flask forms
    - **Utils.py**: user creation and login functions, data retrieval functions
    - **Templates**: UI, HTML templates - rendered by Flask, page data handled by Jinja2
    - **Static**: CSS files and images
    - **App.py**: Flask application, routes (HTTP requests, data management, models interaction, API)
    - **README.md**: requirements, instruction, references

- **Architecture**

## Database:

The SQL Script contains tables for Users, Tags, Tasks, ArchivedTasks, Goals, as well as Hobbies, Playlists and TaskOwner tables for future application improvements. ArchivedTasks is a copy of Tasks table and is used to store tasks marked as completed in Tasks table. Database uses the below operations for data manipulation:

- archive_tasks trigger - inserts tasks marked as completed into ArchivedTasks table
- delete_tasks() stored procedure - removes completed tasks from Tasks table once inserted into ArchivedTasks table
- remove_completed event - calls delete_tasks()

## Flask Application:

Data added and updated by users via the UI rendered by Flask. Flask endpoints handle data submission and prepare it for database operations. SQLAlchemy translates data into SQL commands executed by endpoints to add, update or retrieve data to/from the database. Data is sent back to Flask and rendered in the UI. The application and data are protected by CSRF, authentication and session management as well as error handling and form validations.

## User Creation and Login

### Introduction

This will provide you with an in-depth understanding of the user creation, login and testing process used in this application.

### User Creation

The user creation feature allows new users to create their system accounts. This section will focus on the process of adding new users.

**Method:**

1. To locate the user creation page make sure the server is running enter http://127.0.0.1:5000 into the browser, and press the hyperlink to create a new account, (create one **here)**. This will redirect you straight to the login page.

2. Fill in all the necessary credentials, name, username and password. Where, the name is your full name (or nickname), a valid username which has not already been taken, and a valid password.

3. Once completed, submit the form by pressing the ‘Sign Up’ button. If the credentials you have entered are correct, you should be redirected to the home page and your credentials stored in the database.

**Troubleshooting:**

- Duplicate username: If the username has already been taken, an error message will appear and should prompt the user to create a new username.

## Login

The login functionality allows users to securely access the application. This section outlines the steps to authenticate users via the login process.

**Method:**

1. Ensure the server is running- the login page can be accessed by navigating to http://127.0.0.1:5000.
2. Users will then be prompted to enter their **username** and **password** in the respective fields.
3. After entering the credentials, click the "Login" button to submit the form.
4. Upon successful login, users should be redirected to the dashboard or homepage. If the credentials are incorrect, the fields will be refreshed.

**Troubleshooting:**

- Invalid credentials: If the username and/or passwords are incorrect, the signup form will refresh

## Password security

To ensure that passwords are stored in the database securely, the library ‘Werkzeug’ and specifically the Security library have been implemented.

**Method:**

- Upon user creation, salt is added to the associating password and is then hashed (using the PBKDF2 with SHA-256 algorithm) to avoid the possibility of rainbow table attacks. This is automatically done once the signup form has been submitted, by using the ‘generate_password_hash’ function. The hashed password is then stored in the database securely. In order to authenticate the user during login, the function ‘check_password_hash’ is used which extracts the hashing method, salt and the hashed password. It then uses the extracted salt and method to hash the password the same way it was hashed originally. It then compares this newly generated hash with the one stored in the database to see if they match, if so, the password was authenticated successfully, otherwise, the login attempt will fail.

**Spotipy API**

- **Overview -** 'Spotipy is a lightweight Python library for the Spotify Web API. With Spotipy, you get full access to all of the music data provided by the Spotify platform.' - *Reference*: *Welcome to Spotipy!* (no date) *Welcome to Spotipy! - Spotipy 2.0 documentation*. *Available at: https://spotipy.readthedocs.io/en/2.24.0/# (Accessed: 24 August 2024).*
  It has been used in multiple ways: to search for 2 artist's albums and get 3 different playlists per each endpoint/task type. They return urls that the user can select and it will open that specific playlist/album in Spotify in a new tab in the browser.

- **API Authentication -** In order for Spotipy to authenticate and connect to Spotify, we have to use the OAuth authorization code flow. A Flask session is set up and the access token is cached. Then an instance of the Spotify client is created which uses the token (and refreshes it automatically), which allows the app to interact with the Spotify Web app.

- **API Endpoints Used:**

  - Searching for a specific artist's albums and retrieving urls and names
  - Retrieving playlist details (name, urls).

- **Data Parsing -** When the user selects a tag and clicks 'GO' a GET request is sent to the Spotify API to return all available data (items) within the given Spotify URI(s). From here, we only select the fields that we want to display to the end user in a HTML list (<li></li>) format.

- **Error handling:** The user needs to have logged into Spotify in order to proceed to the home page. In the event that the user cannot access the Spotify login page and a 500 error code is returned, we handle this by redirecting the user to an error page in HTML to let them know that there is an error with the Spotify Server.

- Should the user successfully reach the Spotify login page, error handling for the login process is handled by Spotify.

- Now the user has successfully logged into Spotify, they will be redirected to an HTML page where the user is able to select a tag. When a user selects a tag and clicks 'GO', they are redirected to the URL according to each tag. If there is an issue with Spotify, an error message will be returned from Spotify (not handled by the application directly).

- The only other action a user can perform on this page is to click the 'LOGOUT' button. When a user clicks the logout button, the session is cleared and they are redirected to the Spotify login page. Should there be an issue with accessing the Spotify login page, this error will be handled by Spotify.

## 4. Implementation and Execution

- **Team member roles:**
    - **Alicja Nazaniny** - user login, testing, integration
    - **Cherryl Sinclair** - API(not used), database for pov on API not used, documentation inc Readme, code evaluation
    - **Ekaterina Bluvshtein** - database, app, front-end, testing, integration
    - **Gracie Fenemer** - Spotipy API, front-end, testing, integration
    - **Lidia Sarata** - database, documentation
    - **Mary-Ann Okereke** - database, exception handling, slides
- **Tools and Libraries Used:**
    - **GitHub** - coding contributions, reviews
    - **Flask** – web framework
    - **Jinja2** - templates
    - **SQLAlchemy** – database interactions
    - **MySQL** – database management
    - **Flask - Bootstrap** – CSS framework
    - **Flask-WTF** – Form handling, CSRF protection
    - **Flask session** - session management
    - **Spotipy -** Python API for Spotify
    - **Werkzeug - Security** - Password Hashing
- **Implementation process**
    - **Achievements:**
        - Successful API integration,
        - Session protection,
        - Interactive front-end
    - **Challenges:**
        - Multiple user handling
        - Session handling
        - Data integrity
        - API integration and limitations
        - Time constraints, integrations, pair programming
    - **Changes:**
        - Database adjusted in development
        - Did not implement task viewing by weeks/month

- **Agile development:**
    - Regular revisions and updates
    - Peer reviews
    - Code refactoring

## 5. Testing and Evaluation

- **Testing strategy**
    - Unit testing
    - Integration testing
    - User testing

- **Functional and user testing**
    - Endpoint tests
    - Database manipulation tests
    - Form submission tests
    - HTTP response tests
    - User signup and login tests
        - login page set-up, login success, signup success, sign-up failure, login redirection, logout success tests
    - API routes tests

- **System limitations**
    - Tasks created as completed are not archived automatically and need to be updated before archiving
    - Goal owner has to a username
    - Users are only able to listen to predefined playlists

## 6. Conclusion

In conclusion, the project successfully met its objectives and exceeded the requirements for a minimum viable product. "The Manager" is a creative solution for task and goal management with user friendly features and responsive user interface. The project successfully implements key functionalities with a focus on user experience, seamlessly interacts with a database, allows multiple users to use the application and includes external API integration. Given more time, future enhancements could include playlists configuration, rewards, event finder. Overall, the project delivered a functional application whilst adhering to coding best practices.