

Habit Tracking App

Python Application with DB back-end.

Made by Tom S.

https://github.com/LeftEmpty/habit_tracker/

Table of Contents

1. Overview
 - Features
 - Structure
 - Data Flow
2. Usage
 - Usage instructions mirrored from GitHub repository
3. Database
 - Table Structure - ER Diagram
 - Request to Controller - Example
4. Back-end
 - Object Classes & Request Handler
5. Front-end
 - GUI Class Setup
 - Usage

Chapters 3 through 6 explain how the project / code works. They don't touch the project's content or usage and could therefore be ignored. I strongly recommend reading the database chapter.

Overview

- Features
- Structure
- Data Flow

Features

- App features include:
 - Graphical User Interface (GUI)
 - User login / register
 - Habit creation, editing, and deletion
 - Public & Private Habits (explained further on following slides)
 - User & Habit statistics
 - SQLite3 Database controller (incl. Logger & Test Data)
 - Test unit
- Usage is clearly explained in the GitHub repository's *[Readme!](https://github.com/LeftEmpty/habit_tracker/?tab=readme-ov-file#usage-instructions)

*https://github.com/LeftEmpty/habit_tracker/?tab=readme-ov-file#usage-instructions

Structure

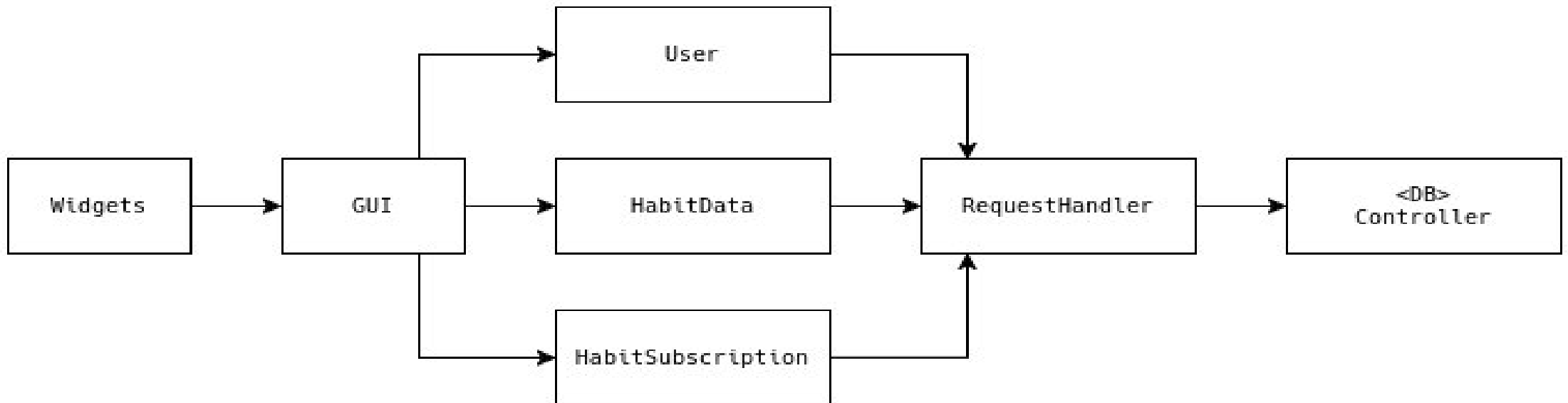
The following showcases and explains the overall (folder) structure of the project.

- GUI
 - Consists of the main class, screens & widgets which include popup windows.
- Obj (Objects)
 - User based system.
 - Users own Subscriptions, actions/requests use the users id.
 - The user is also set as `cur_user` in the GUI on successful login)
 - Separated Habits and Subscriptions to support private and public habits.
 - A Habit Subscription is based on a Habit Data and contains additional information about periodicity and streaks.
 - The end-user only sees habits, the name 'subscription' is only used in-code.
 - Request handler acts as a sort of API that interfaces object classes and the database controller.
 - This should enable quick and easy adjustment or replacement of the database to support servers, different systems, etc.
- DB
 - Database controller containing SQL queries.
 - Also contains the script that populates the database with test data (adjusts to date) and a logger.

Data Flow Overview

The GUI takes in user input. The GUI itself may also use the currently logged in user to request a list of active habit subscriptions for example.

These actions/requests reach the request handler via the appropriate object class. The request handler acts as some sort of API for the classes to make requests to the database controller. The request handler finally formats the query result and returns it.



Usage

- Features
- Structure
- Data Flow

Usage

- I recommend reading the usage instructions via the GitHub's Readme, mirrored here for content completion.
- The following 4 slides will explain the usage instructions.
- The chapters after this one include more detailed information about how the project works.

Launch the App & Register or Login

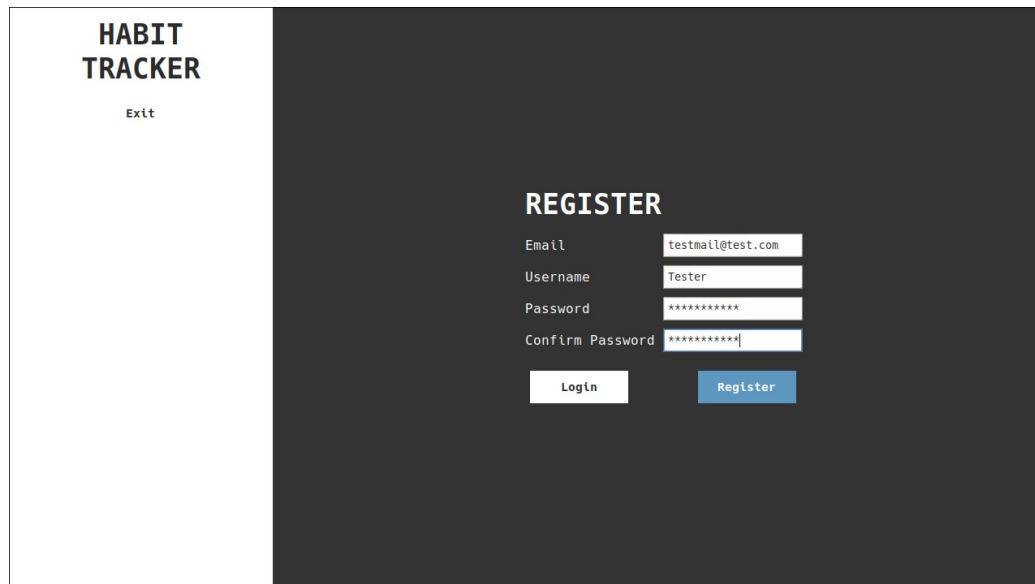
Launch the application

- Installation guide in repository's Readme.
- Use the --debug argument to utilize test data.

Log in or register a user

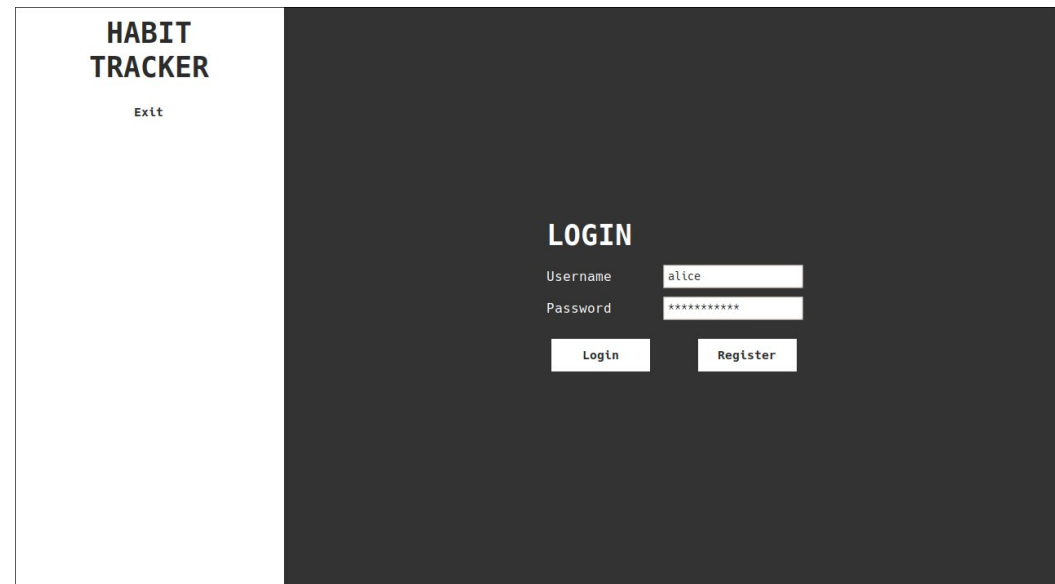
- Users alice & bob are provided when using the test data.
- Password for both is simply 'password123'.
- The login information is also shown in the Terminal log when using the --debug argument.
- The Login/Register buttons swap the screen when on the 'wrong' screen or fulfill the action when already on the 'correct' screen.

Image: Register Screen



The screenshot shows the 'HABIT TRACKER' app interface. On the left, a white sidebar contains the text 'HABIT TRACKER' and 'Exit'. The main area is dark gray and displays a 'REGISTER' form. The form includes four input fields: 'Email' (containing 'testmail@test.com'), 'Username' (containing 'Tester'), 'Password' (containing '*****'), and 'Confirm Password' (containing '*****'). Below the fields are two buttons: a white 'Login' button and a blue 'Register' button.

Image: Login Screen



The screenshot shows the 'HABIT TRACKER' app interface. On the left, a white sidebar contains the text 'HABIT TRACKER' and 'Exit'. The main area is dark gray and displays a 'LOGIN' form. The form includes two input fields: 'Username' (containing 'alice') and 'Password' (containing '*****'). Below the fields are two buttons: a white 'Login' button and a white 'Register' button.

Create or Add Habits

- On the Home screen you can view habits that are due today.
- Here you can also 'Create' or 'Add' new habits by clicking one of the buttons.
- Creating a new Habit will open a popup, selecting a periodicity and providing the required input will let you click the 'Create' button in the popup and the habit will be created and automatically subscribed to.
- Adding a new habit will lead you to the public habits screen from which you can subscribe to a habit by selecting a periodicity and clicking the [+] button on the listed habit widget.

Image: Home Screen

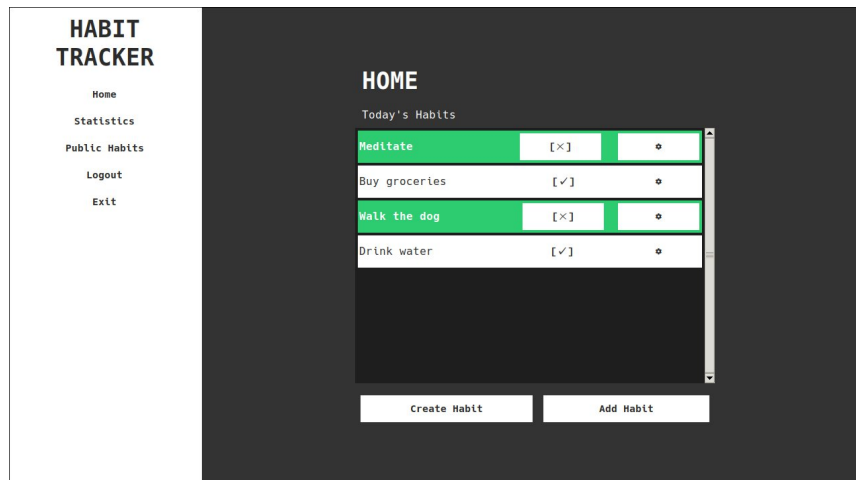


Image: Create Habit Popup

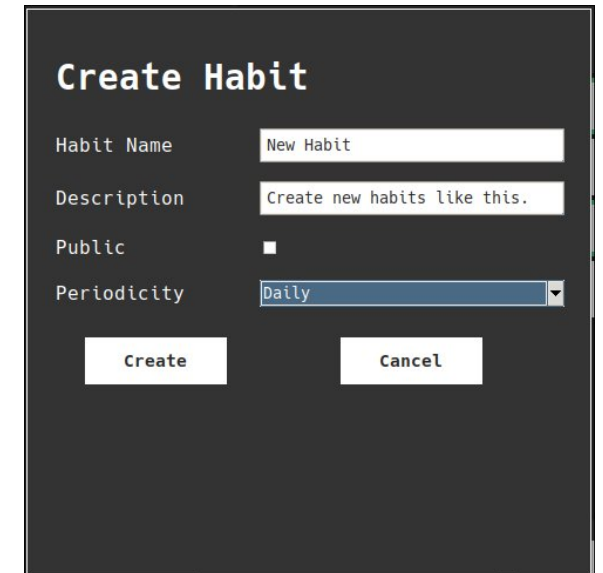
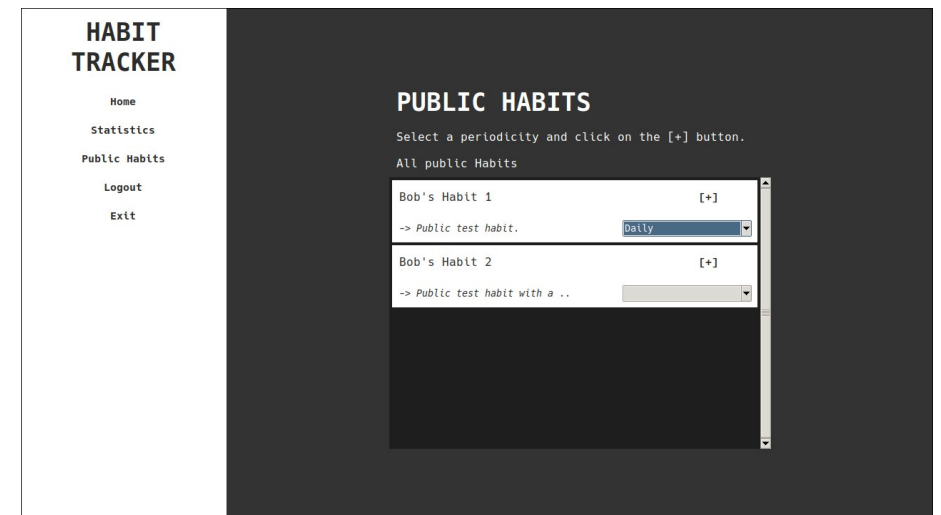


Image: Public Habits Screen



Complete Habits & Edit Habits

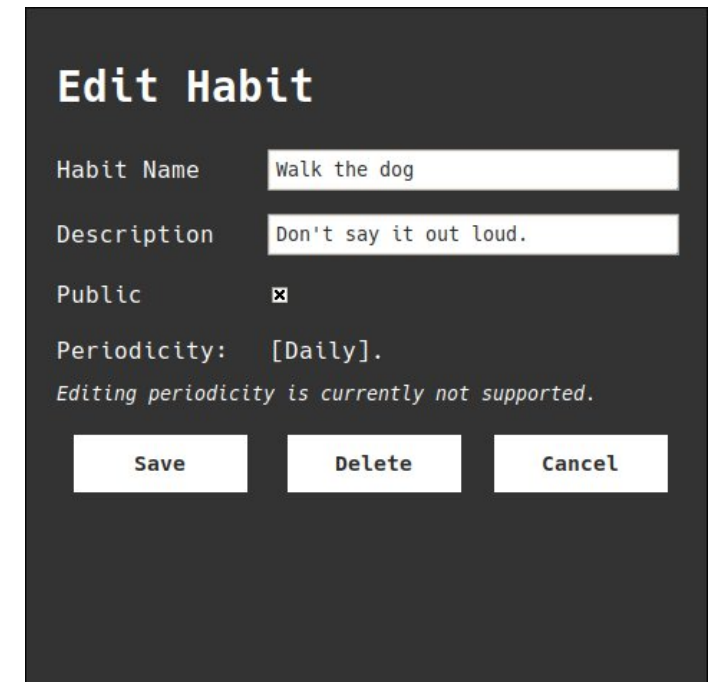
Mark habits as completed

- On the Home screen (see above for image) you can mark your habits as done, by clicking the [✓] button on one of your listed subscribed habits.
- Can be undone via the [X] button that's shown on completed habits.
- This will create/destroy a completion for the Habit Subscription.

Edit habits as necessary

- Clicking the ⚙ button on a subscribed habit enables the user to edit the habit.
- The habit data entry is then modified with the newly input data.
- This propagates to all other users as well.
- As of now, only the author user can edit the habit, and the habit data database entry is then modified accordingly.
- There is currently no support for the periodicity setting.

Image: Edit Habit



The image shows a dark-themed modal window titled "Edit Habit". It contains several input fields and a checkbox. The "Habit Name" field contains "Walk the dog". The "Description" field contains "Don't say it out loud.". The "Public" checkbox is checked. The "Periodicity" field is set to "[Daily]". Below this, a note in italics states "Editing periodicity is currently not supported.". At the bottom, there are three buttons: "Save", "Delete", and "Cancel".

Edit Habit

Habit Name

Description

Public ☒

Periodicity:

Editing periodicity is currently not supported.

View statistics and track streaks & progress over time

- On the Statistics screen, the user can view their overall Statistics, as well as view a list of all their subscribed habits ordered by periodicity via the List Habits by Periodicity button (Opens popup window).
- Habit specific stats can be viewed by selecting a habit in the drop-down and clicking the Show Habit Stats button. This will update the screen to reflect the selected habits stats.
- The Show Default Stats button can be used to view the general user stats again, this will also be shown when re-opening the screen.

Image: Default Statistics

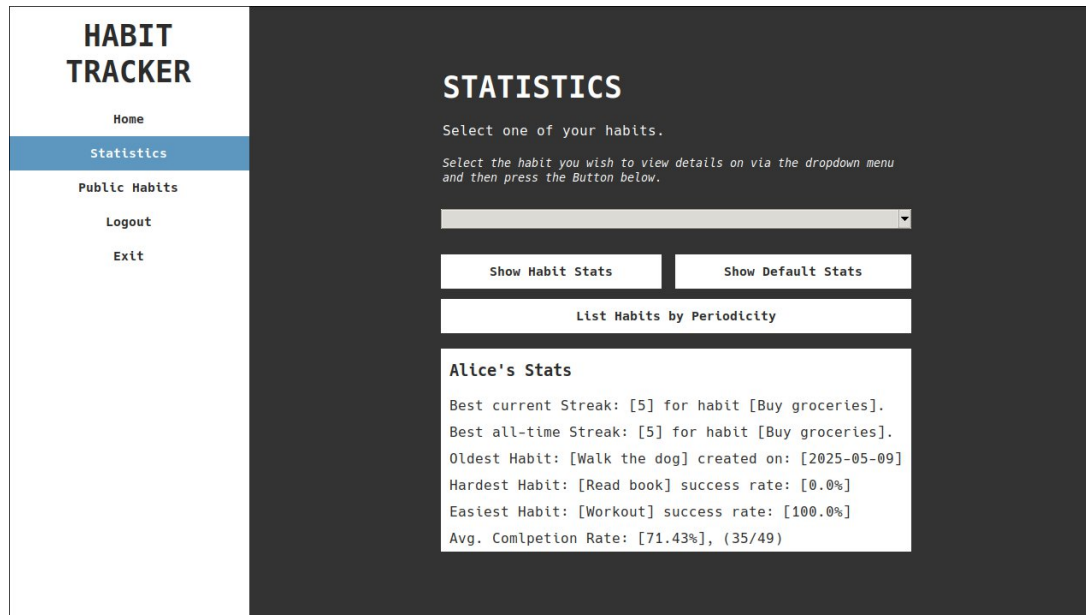


Image: Habit Statistics

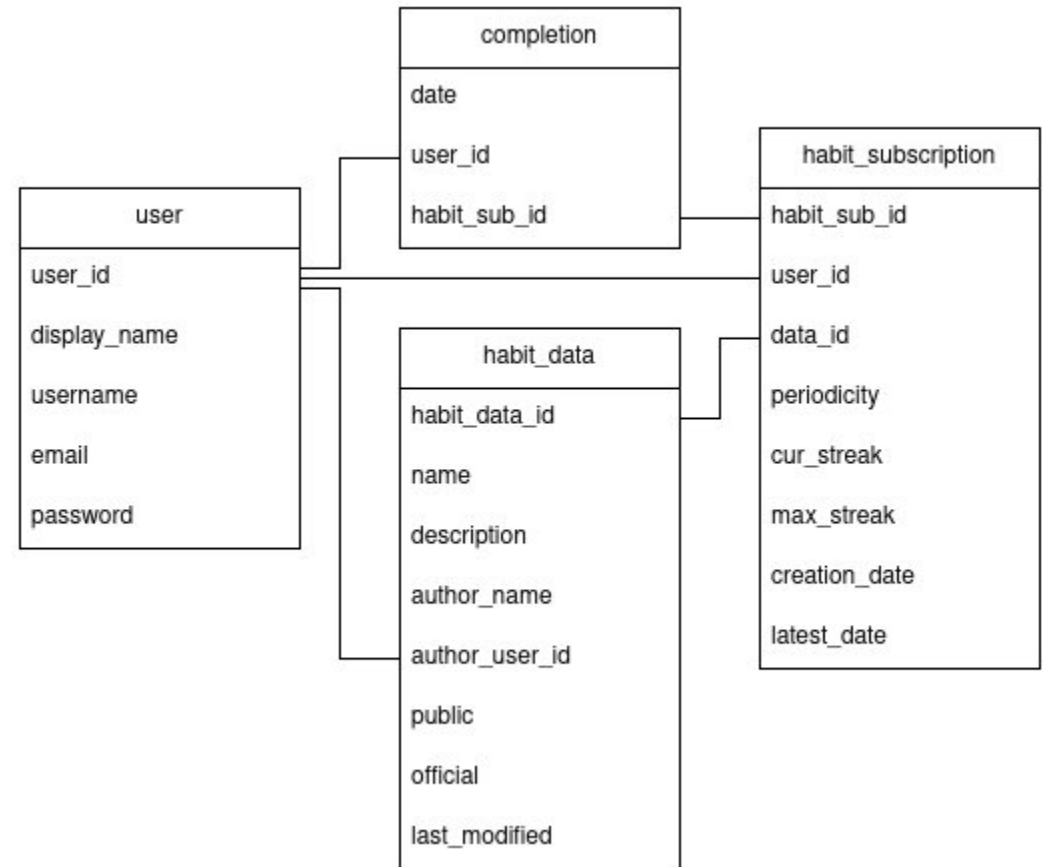


Database

- Table Structure – ER Diagram
- SQL Query Example
- Request to Controller Example

Table Structure - ER diagram

- Subscriptions serve to 'connect' habit data with users
- When a user 'checks off', i.e. completes a habit (via GUI), a new completion entry will be created.
- Completions entries can be used to calculate stats for a given subscription and user.
 - There are no 'failed completions' the success rate is derived by calculating expected completions and the actual completions retrieved via a query.



Request to Controller - Example

Example of a request chain, user → request_handler → db_controller.

User

```
subs:list[HabitSubscription] = request.get_subs_for_user(self.user_id)
result:list[HabitSubscription] = []

if cond == HabitQueryCondition.ALL:
    return subs
if cond == HabitQueryCondition.RELEVANT_TODAY:
    for s in subs:
        if s._periodicty_relevant_today():
            result.append(s)
```

Database Controller (dbc)

```
try:
    result = cx.execute(
        """
        SELECT * FROM habit_subscription WHERE user_id = ?
        """,
        str(user_id)
    )
    return result.fetchall()
except sqlite3.Error as e:
    log.error(f"Error getting subscriptions from user with ID of {user_id}: {e}")
    return []
finally:
    cx.commit()
    cx.close()
```

Request Handler (request)

```
# query db
results = dbc.db_get_subs_for_user(user_id, conn)
if len(results) <= 0: return []

# format query results
subs:list[HabitSubscription] = []
for r in results:
    sub = HabitSubscription(
        user_id=r[1],
        data_id=r[2],
        periodicity=r[3],
        cur_streak=r[4],
        max_streak=r[5],
        creation_date=r[6],
        last_completed_date=r[7],
        sub_id=r[0]
    )
    if sub:
        subs.append(sub)

return subs
```

- Queries use try blocks and include error logging.
- Input is not inserted directly to avoid injection attacks.
- DB Controller only does it's job, does not format anything.

Back-end

- Object Classes & Request Handler

Object Classes

- Object classes, as mentioned, consist of Users, HabitData, HabitSubscription (also contains Completion and Periodicity Enum)
- The “request_handler.py” file is also in the obj folder but follows a functional programming paradigm allowing the aforementioned classes to use it’s functions as necessary.

Front-end

- GUI Class Setup

GUI Class Setup

- The main GUI class stores the current user and theme (Tkinter style). It is mainly responsible for displaying the base layout and populating the sidebar.
- Screen classes (base-class & individual classes) are instantiated on app launch and receive data in the `on_screen_opened` class. Opening a screen via the sidebar 'opens' it in the `content_area` of the base layout in the GUI class.
- `Widgets.py` contains a lot of useful widgets and popup windows used by the screens.