OneTouch Design Doc

CSCI 492 Senior Project

Noah Stull
Will Snyder
Andrew Pasimio
Wylie Mickelson

December 8, 2025

Contents

1 Product Overview **2**
1.1 Background . . . 2
1.2 Major Features . . . 2

2 Implementation Progress **2**
2.1 Conclusion of CSCI 491 . . . 2
2.2 CSCI 492 Progress . . . 2

3 Project Architecture and Design **3**
3.1 Architecture . . . 3
3.2 Dependencies . . . 3
3.3 UI Design . . . 4
3.4 Testing and Verification . . . 9

4 Data Layer **10**
4.1 Data Management . . . 10
4.2 Saved Object Structures . . . 10
4.3 Local Storage . . . 11

5 Data Exportation **11**

1 Product Overview

1.1 Background

The OneTouch app is designed to optimize the note-taking process for its users. OneTouch is tailored to professionals and employees with narrow time frames and limited attention. OneTouch provides customizable note formats to simplify data recording for workers on the job. The application requires only a single button click per note to make recording easy and fast.

1.2 Major Features

1. Customizable note outline creation and editing.
2. Note-taking based on a predefined and saved outline.
3. One-handed accessibility for ease of use.
4. Note and outline organization.
5. Export to pdf service for completed notes.

2 Implementation Progress

2.1 Conclusion of CSCI 491

After 491 ended, we had a good list of features that we wanted to build into our app and a solid idea of how we wanted it to look. The requested features from our client, Jackie, were somewhat underwhelming, so we forced a bit of scope creep into our project. For our deliverables, we had produced a design document, PR/FAQ, and some Gherkin tests that we wanted to implement. All of these first quarter documents can be found inside the github repo under FinalSubmission/491.

2.2 CSCI 492 Progress

- **Andrew's Progress:** Object structure and design, session summary page, and data exportation.
- **Wylie's Progress:** Local storage implementation and linking template/session creation, deletion, and editing between the storage and UI.
- **Noah's Progress:** Note session page, creation of mockup template based on client's data, note and template objects.
- **Will's Progress:** Test creation, including unit, widget, and integration tests for both note.dart and template.dart files.

3 Project Architecture and Design

3.1 Architecture

The architecture can be split into two layers: a UI layer and a Logic/Data layer. The UI layer will be responsible for the user view, inputs, and interactions. The Logic layer will be responsible for logic, use cases, pulling data from local storage, and passing information on to the View layer. The following diagram (Figure 1) represents a high-level outline of this architecture.

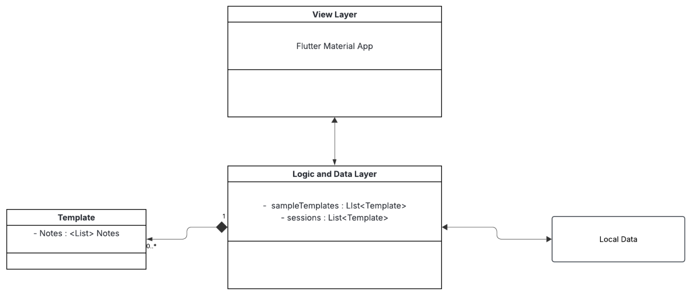


Figure 1: System Architecture

The interactions between the UI and Logic layer can be see in the diagram (Figure 2). Here, the pages-related classes interact with Template and Note components to display and save the proper data.

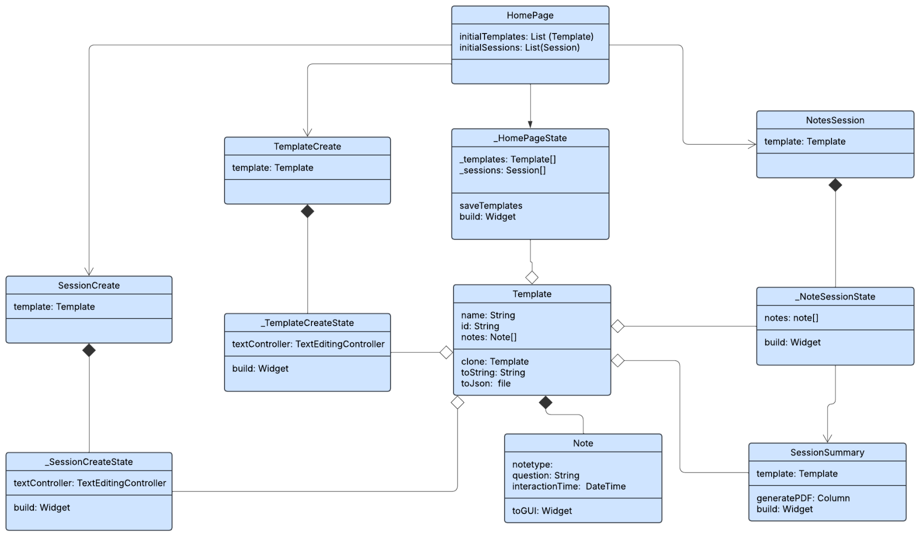


Figure 2: Class Diagram

3.2 Dependencies

- **Dart** - Main language
- **Flutter** - Cross-Platform Framework for Android/IOS apps

3.3 UI Design

Home Screen This is the main screen that the user will see when opening the app for the first time. Recent sessions will be stored and ready to go for the user to return to, and they are able to start new sessions and create new templates. (Figure 3).

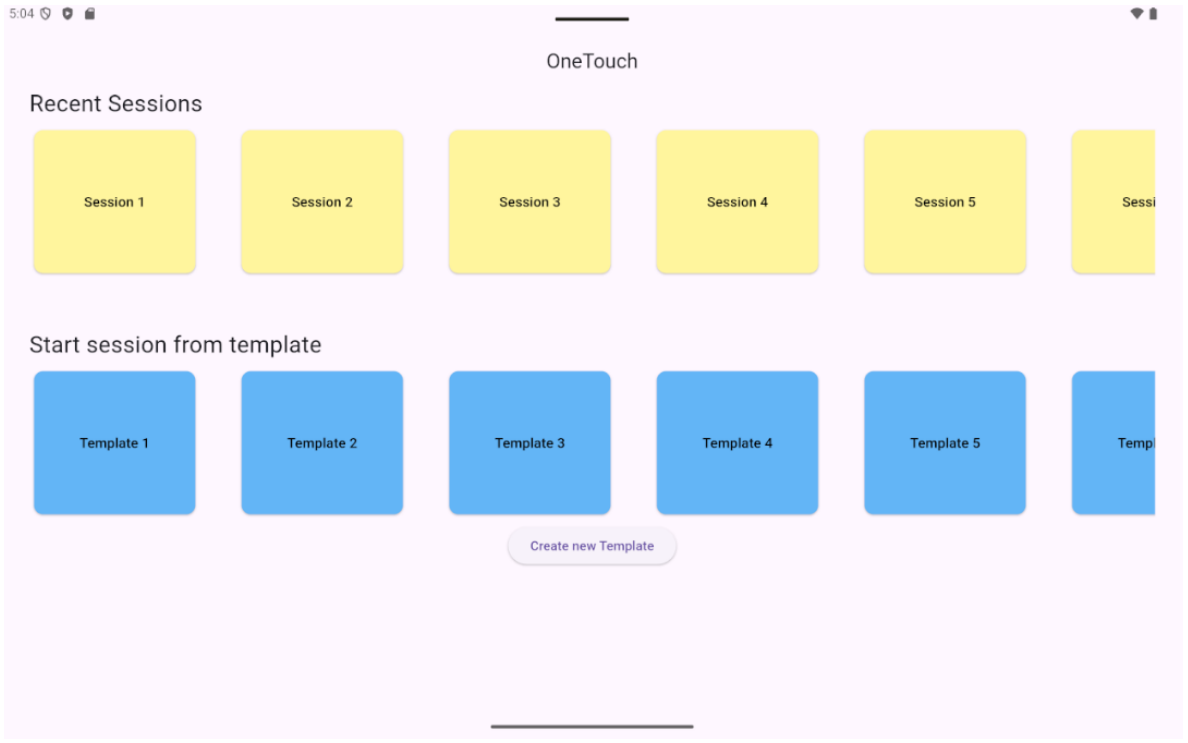


Figure 3: Home Screen

Session Create Screen Upon clicking a saved template from the list on the home screen, a new page will pop up which will ask the user to add a name for the new session they want to create. After confirming, it will immediately begin the new session and go to the Note Session page. (Figure 4).

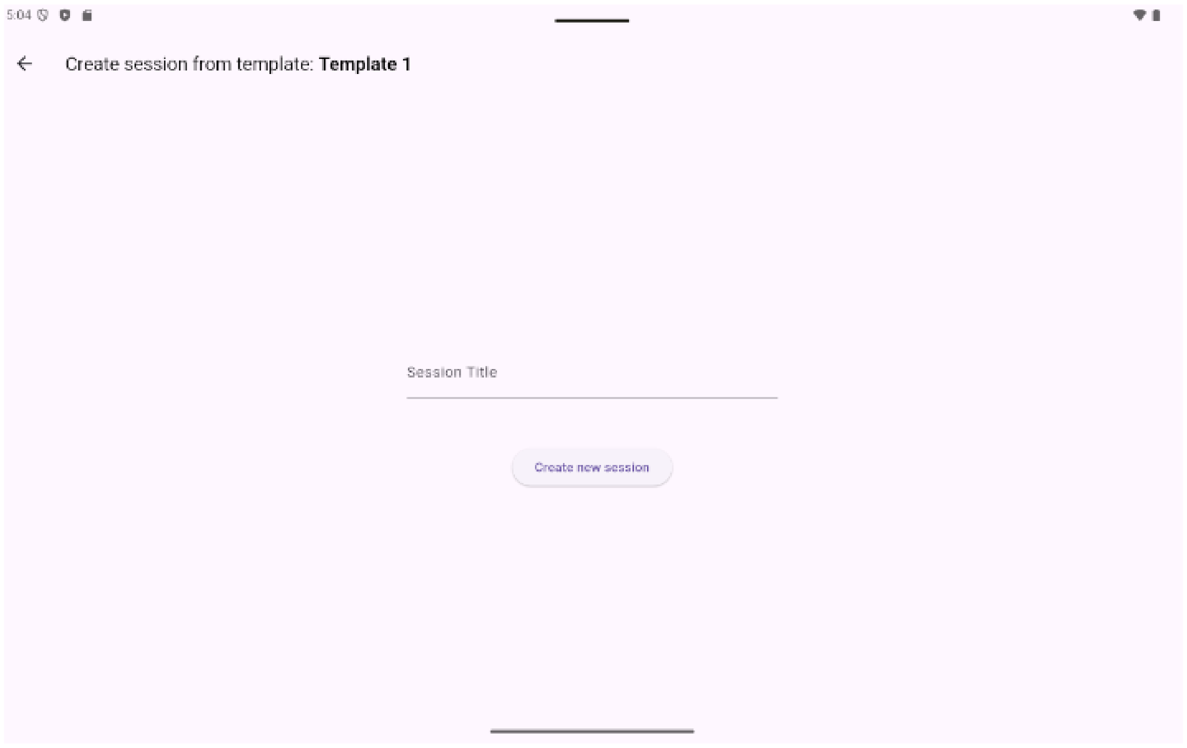


Figure 4: Session Create Screen

Template Create Screen After choosing 'Create new Template', the user will see a page that closely resembles the Note Session screen, but modified to alter the overall template. This is purposely designed so that the user can see exactly how the template will be shown when actually filling it out. (Figure 5).

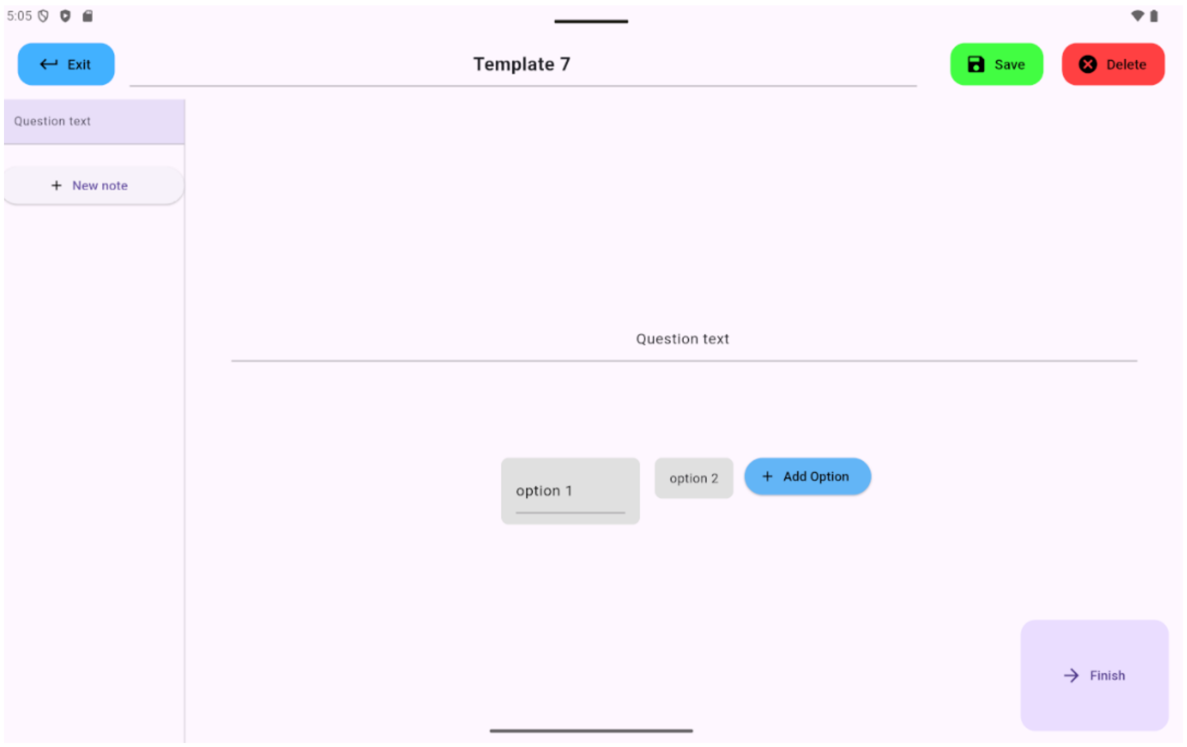
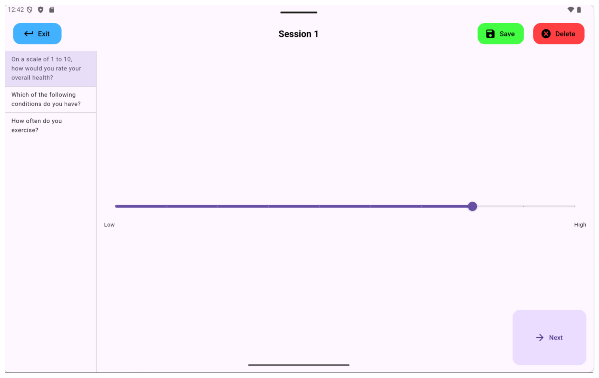


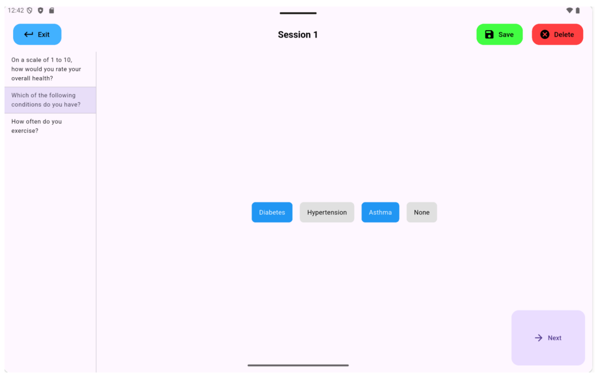
Figure 5: Template Create Screen

Note Session An example of a note-taking session using a predefined template. Each section contains a different kind of note: (Number Scale: Figure 6), (Multiple Choice: Figure 7), and (Single Choice: Figure 8). After a user is done with the note they will press the arrow to move to the next note section. At any point, the user can exit without saving, save the current progress, or delete the session by pressing the corresponding button.



12:42
Exit
Session 1
Save
Delete
On a scale of 1 to 10, how would you rate your overall health?
Which of the following conditions do you have?
How often do you exercise?
Low
High
Next

Figure 6: Number Scale Note



12:42
Exit
Session 1
Save
Delete
On a scale of 1 to 10, how would you rate your overall health?
Which of the following conditions do you have?
How often do you exercise?
Diabetes
Hypertension
Asthma
None
Next

Figure 7: Multiple Choice Note

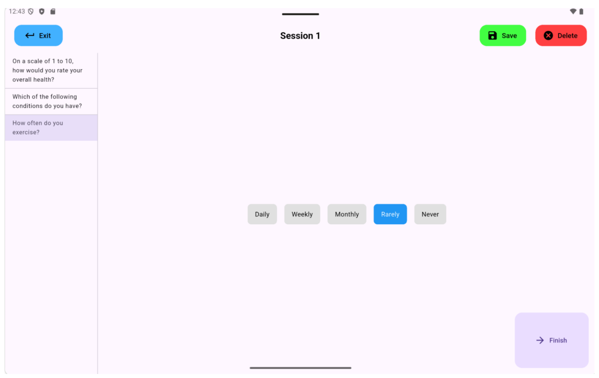
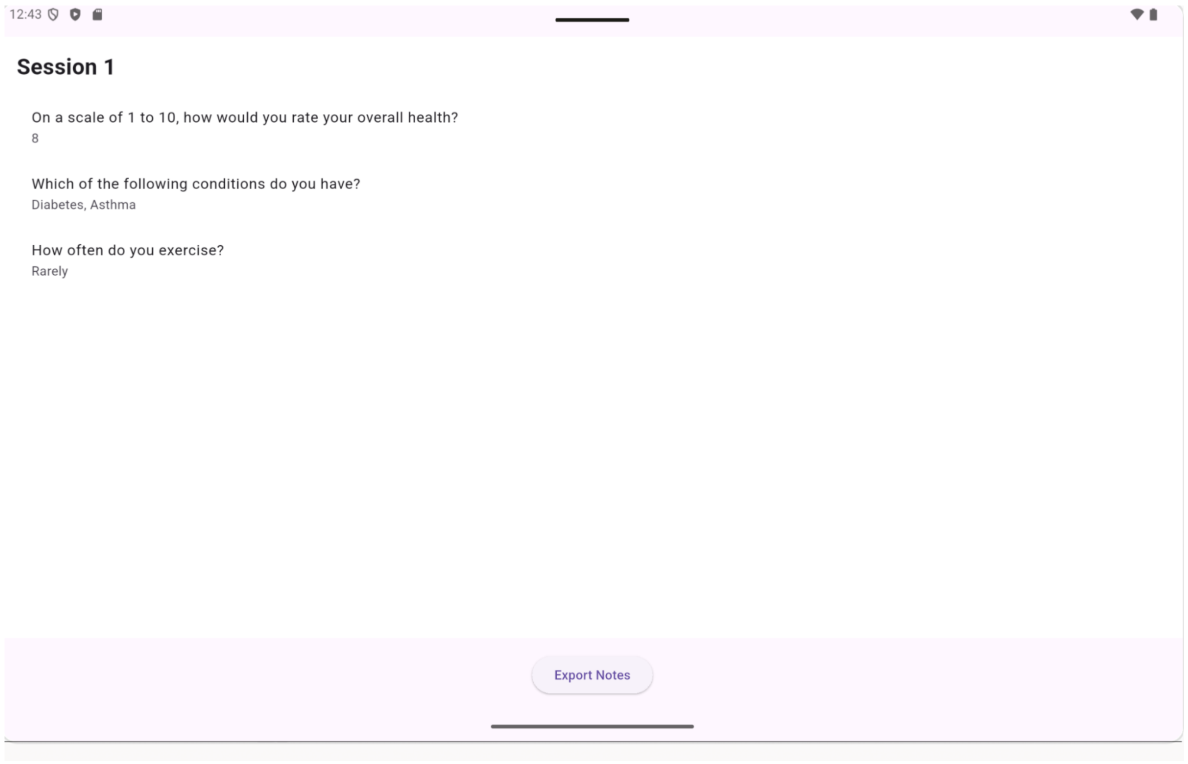

12:43
Exit
Session 1
Save
Delete
On a scale of 1 to 10, how would you rate your overall health?
Which of the following conditions do you have?
How often do you exercise?
Daily
Weekly
Monthly
Rarely
Never
Finish

Figure 8: Single Choice Note

Session Summary Screen If at any moment the user wants to conclude early, they can press the "finish" button in the top right and conclude early. This, along with normal progression, will take you to the results screen. (Figure 9)

The final screen displays all of the details that the user input in a bulleted list, along with their corresponding time stamps. The buttons on the bottom allow the user to go back and modify data again, save and return to the home screen (Figure 3), or export via email. Upon saving, the recent sessions will update to keep the new note in the first slot until a new one is written. Choosing to mail the document will convert the results page into an equivalent pdf to be sent to the recipient. The format of this sheet will depend on how the user customized the trials and session. (Figure 9)


12:43

Session 1

On a scale of 1 to 10, how would you rate your overall health?
8

Which of the following conditions do you have?
Diabetes, Asthma

How often do you exercise?
Rarely

Export Notes

Figure 9: Session Summary Screen

3.4 Testing and Verification

Swift Testing will be used for unit testing the logic and data layers of the app. Some unit tests that will be implemented are as follows:

Updates since 491:

- note_test.dart added: This file contains unit, widget, and integration tests.
  - Starting off with the unit tests, there are tests for the NumberNoteScale, MultipleChoiceNote, and SingleChoiceNote. NumberNoteScale tests include value tracking (getValueString), interaction tracking (markInteraction), and serialization, while the two ChoiceNotes do similar tests for getValueString() and both check for empty states.
  - Widget tests include checking the same three classes. NumberNoteScale should check component rendering and UI structure, MultipleChoiceNote options display and selection mechanics/restraints, and SingleChoiceNote checks exclusive selection and deselection.
  - Integration tests combine some of this to test a complete workflow. They include interaction and serialization, and a full round-trip (create note, render UI, interaction, serialize/deserialize, data integrity).
- template_test.dart added: This file tests unit tests in two different sections and also has integration tests. The first unit test section is core functionality is that of the constructor for the template.dart file as well as state management in 5 different methods (setname, add, remove, getAt, clear). The second is to do with serialization, and that includes JSON conversion (toJson, fromJson), type-specific serialization, and cloning as well. The integration tests come for I/O operations like saveTemplateData and getTemplateData along with checking if full persistence exists.

Focus for 493:

- Object tests are done unless we need to add another file, so the shift for this session will be toward implementing tests for the lib/Pages section. This includes files for the home page, note session, creating a session, the session summary, and creating a template. This is currently about double the work of what came in lib/Objects, but shouldn't necessarily take twice as long given we have a format for how we should write these moving forward.

4 Data Layer

4.1 Data Management

Due to HIPAA compliance, no data will be sent or stored on any remote servers. Instead, all notes and models will be saved directly to the hosting device. Therefore, the "backend" will not be hosted on a server itself. It will be part of the same package that contains the other layers. To accomplish this, we will have a backend data management class containing all relevant data retrieval and storage operations, which will act as a kind of "mock" backend.

Flutter has built in serialization methods that allow for a simple toJson and fromJson for storing and retrieving data.

4.2 Saved Object Structures

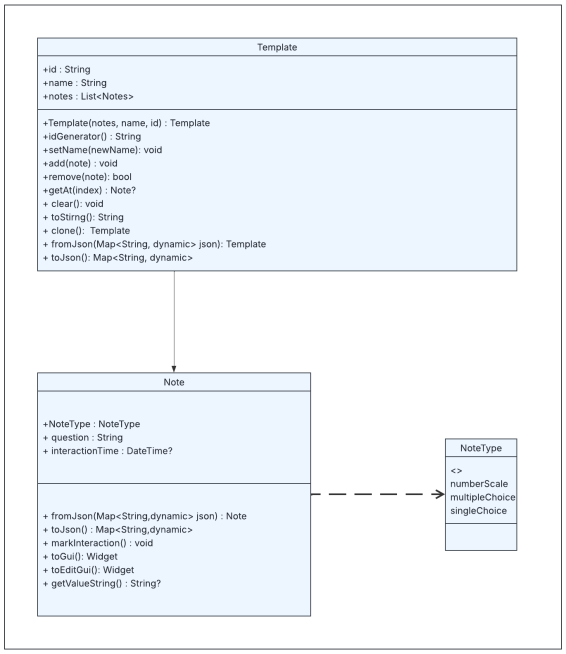


Figure 10: Data Structures

The structure of the saved objects can be seen in Figure 10, with a detailed description of each as follows:

- Template
  - The template class is utilized in two ways, first to build a blank empty template and secondly to fill it out with responses to each note in the template. The app supports multiple instances of a template being filled out with different answers and each of these versions can be returned to at any time.
  - The toJson and fromJson functions allow for easy storage and retrieval of the template
  - The clone functionality makes it easy to create new Template objects from empty Templates or to make copies of filled out ones that need to be updated

- Note
  - The interactionTime is a nullable DateTime object that allows for easy sorting of the parent template and allows for logging of what previous changes have been made
  - The toGui() function allows for a quick custom Widget creation that can be used anywhere
  - Each NoteType has it's own toGui and getValueString functions to allow for the different input/output types.

4.3 Local Storage

There isn't a need for overly complex data storage in this application, so we decided to use JSON files to store our application's data. At the moment, we have two different files, each represented as an array of Template objects:

- templates.json: Stores the Template objects with default placeholder data.
- session.json: Stores Template objects with saved data from the user.

The UI of our application depends on these files to obtain object data. Any time there is a save operation on the UI end, we call the corresponding function to update these files. For development purposes, there is a set of mock data we use in development to make sure the UI is displaying as intended.

5 Data Exportation

Notes will have two options for exportation: save a pdf locally or send a pdf directly to an email (or a set of emails). The sessionSummary page handles the building of the pdf in it's logic layer and then will either send it to the users email or save it locally to a directory. Upon saving the pdf or sending it to an email there will be a snack bar notification. The snack bar will also be used to notify the user of any user end errors (i.e. bad email or directory not found).

The current pdf export functions appears as such:

```
Future<bool> exportPdf() async {
    final directory = await getApplicationDocumentsDirectory();

    final file = File(path);
    final bytes = await pdf.save();
    await file.writeAsBytes(bytes);

    // Print the saved file path to the console for debugging
    // (useful to locate the file on device/emulator)
    print('PDF exported to: $path');

    return true;
  }
```