<Project Name>

Architecture Notebook

There is guidance within this template that appears in a style named InfoBlue. This style has a hidden font attribute that allows you to toggle whether it is visible or hidden in this template. Use the Microsoft® Word® menu **Tools > Options > View > Hidden Text** check box to toggle this setting. There is also an option for printing: **Tools > Options > Print**.

# Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

[Always address Sections 2 through 6 of this template. Other sections are recommended, depending on the amount of novel architecture, the amount of expected maintenance, the skills of the development team, and the importance of other architectural concerns.]

# Architectural goals and philosophy

[Describe the philosophy of the architecture. Identify issues that will drive the philosophy, such as: Will the system be driven by complex deployment concerns, adapting to legacy systems, or performance issues? Does it need to be robust for long-term maintenance?

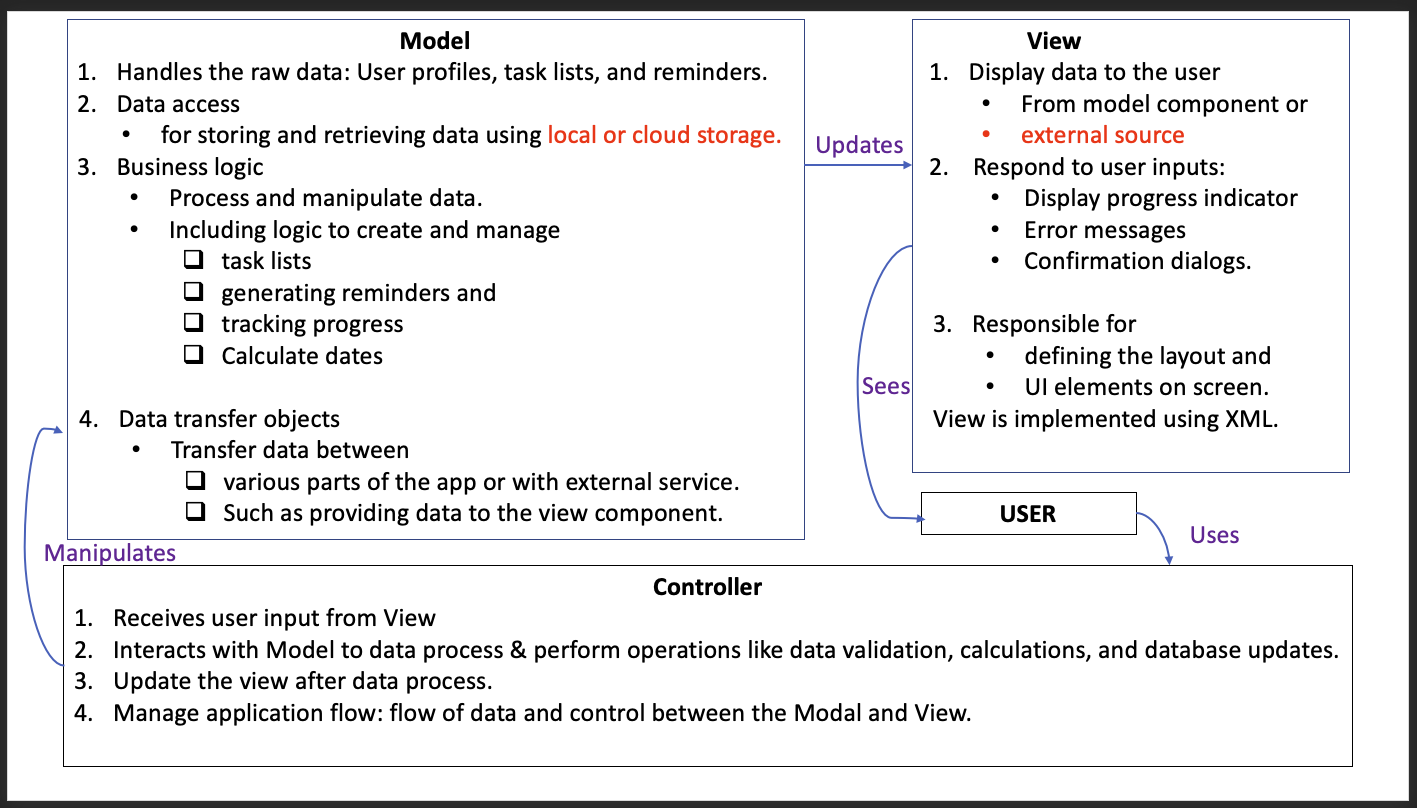
Formulate a set of goals that the architecture needs to meet in its structure and behavior. Identify critical issues that must be addressed by the architecture, such as: Are there hardware dependencies that should be isolated from the rest of the system? Does the system need to function efficiently under unusual conditions?]

# Assumptions and dependencies

[List the assumptions and dependencies that drive architectural decisions. This could include sensitive or critical areas, dependencies on legacy interfaces, the skill and experience of the team, the availability of important resources, and so forth]

# Architecturally significant requirements

To realize the architecture, it is vital to achieve Separation of Concerns (SoCs) to effect scalability, testability, performance, and maintainability. Consequently, 3 components with specific responsibilities will be created.



[reference](https://www.codecademy.com/article/mvc)

# Decisions, constraints, and justifications

The decisions are based on current best practice guidelines for apps in general and for ADHD apps in particular. For instance:

1. Using a modular approach to address separation of concerns.
2. Using Kotlin to write more concise and efficient codes.
3. Thorough testing of the app.
4. Local as well as external storage capability to allow app use without internet.
5. Simple and easy navigation since the app is for ADHD condition.
6. [Colour coding](https://www.fullfabric.com/articles/how-to-design-visual-learning-resources-for-neurodiverse-students) to differentiate tasks or task priorities.
7. Reminders, notifications, and goal settings with flexibility to customising as needed.
8. Gamification to stimulate/maintain motivation via rewards or progress tracking.

Some constraints are:

1. Time and resources to test the app with real users.
2. Access to only free edition of third-party applications, such as Visual Paradigm.
3. Knowledge/experience/competency of team members regarding different platforms such as Android, Kotlin and external/local database available.

Justification

* The best practice approach has proven merits to positively impact the success of a project in the short run (such as debugging) as well as in the long run (scalability or creating a newer version in the future)
* Since the app is for people with ADHD, certain specific features are necessary to make the app useful; such as avoiding cluttering.
* Since certain components require steep learning curve, it is vital to keep things simple as well as manageable to stay within allocated timeframe of the project.
* There are certain limitations to using free editions, such as Visual Paradigm, but there is no genuine need to splash money since we are university students.
* The app is designed to be Android native to allow members to focus on one platform. In addition,
  + Android has the [largest market share](https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/) worldwide.
  + Android has a well-developed ecosystem providing vital features such as frameworks and libraries.
  + Android is open-source and therefore allows greater flexibility to customise codes to suit one’s own needs.
  + Android platform provides integration with other Google products, which provides various APIs and features for app development.

# Architectural Mechanisms

[List the architectural mechanisms and describe the current state of each one. Initially, each mechanism may be only name and a brief description. They will evolve until the mechanism is a collaboration or pattern that can be directly applied to some aspect of the design.]

## Architectural Mechanism 1

[Describe the purpose, attributes, and function of the architectural mechanism.]

## Architectural Mechanism 2

[Describe the purpose, attributes, and function of the architectural mechanism.]

# Key abstractions

[List and briefly describe the key abstractions of the system. This should be a relatively short list of the critical concepts that define the system. The key abstractions will usually translate to the initial analysis classes and important patterns.]

# Layers or architectural framework

[Describe the architectural pattern that you will use or how the architecture will be consistent and uniform. This could be a simple reference to an existing or well-known architectural pattern, such as the Layer framework, a reference to a high-level model of the framework, or a description of how the major system components should be put together.]

# Architectural views

[Describe the architectural views that you will use to describe the software architecture. This illustrates the different perspectives that you will make available to review and to document architectural decisions.]

## Recommended views

* **Logical:** Describes the structure and behavior of architecturally significant portions of the system. This might include the package structure, critical interfaces, important classes and subsystems, and the relationships between these elements. It also includes physical and logical views of persistent data, if persistence will be built into the system. This is a documented subset of the design.
* **Operational:** Describes the physical nodes of the system and the processes, threads, and components that run on those physical nodes. This view isn’t necessary if the system runs in a single process and thread.
* **Use case:** A list or diagram of the use cases that contain architecturally significant requirements.