Software Design
for
Minutes Management System (MMS)

Team 24

G Karthik Balaji
CS19BTECH11001

Rachit Keerti Das
CS19BTECH11034

Gunangad Pal Singh Narula
CS19BTECH11035

Shashank Shanbhag
CS20BTECH11061

Spring 2023

Contents

1 Modules

A module is a self-contained unit of code that performs a specific set of functions. A good module has high cohesion and low coupling. In other words, modules are typically independent of other modules; breaking the program down into this top-down manner is more flexible and allows for easier maintenance, testing, and development.

Module	**Functionality**
UI and User Authentication	The UI provides an interface for the user to interact with the system and perform different activities, such as uploading files or searching for specific queries. It also provides a secure login process that allows users to access the system using their GSuite account. Once authenticated, all authorized users can access the full range of features available to them based on their assigned user role.
Upload and Parsing	Only Admins are authorized to upload documents to the system. When a document is uploaded, the TextEngine parses the document and converts it to a suitable text-based format for storage in the database. The format shall be created so as to preserve sections and various divisions from the original document. If the operation is successful, a success message is displayed to the user.
Searching and Retrieval	The system allows users to search for specific documents by entering relevant keywords. The search engine searches over the document corpus and retrieves the relevant results, which are then sent to the UI for displaying to the user. The SearchEngine will be a PyTorch module, which is a fixed, already-defined class hierarchy by itself, and hence we've not shown it in our class diagram.
Data storage	The database stores all documents in the above-mentioned text-based format. In response to a query, the documents are converted into a text format and sent to the search engine for checking relevant parts. The database stores all documents, such as senate meeting minutes, agendas, and handbooks, thus ensuring that all relevant information is easily accessible to authorized users while preserving necessary section and positioning information.

2 Class Diagram

A class diagram describes the attributes and operations of a class, as well as the constraints imposed on the system.

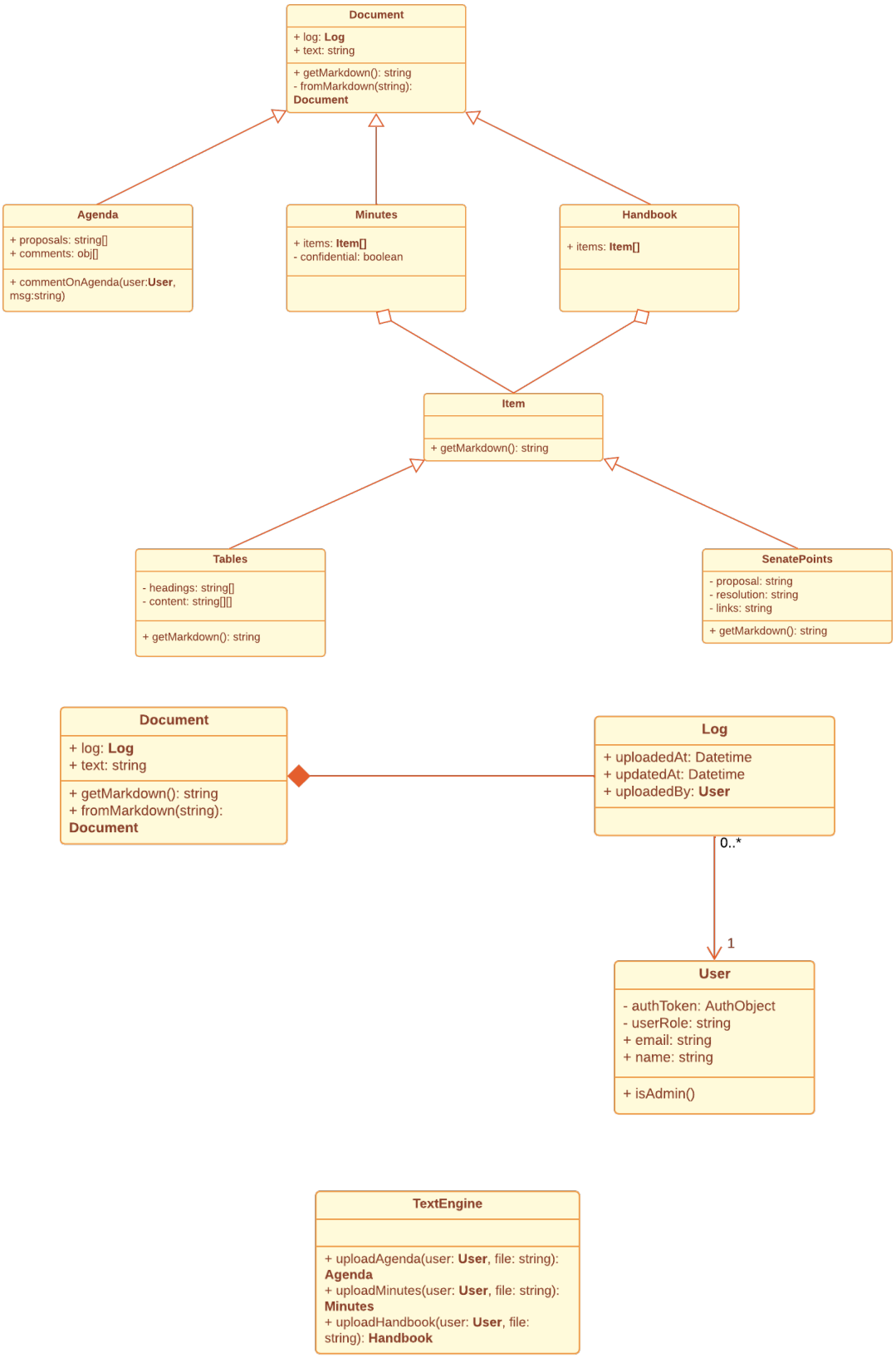

Figure 1: Class Diagram for Minute Management System

3 Sequence Diagram

Sequence diagrams are used to model the behavior of a system and depict the interactions between objects or components in a system over time. In other words, it shows the sequential order of operations in the system for different use cases.

Given below are the sequence diagrams of two main use cases that were provided in our SRS document:

3.1 Use Case 1: Search

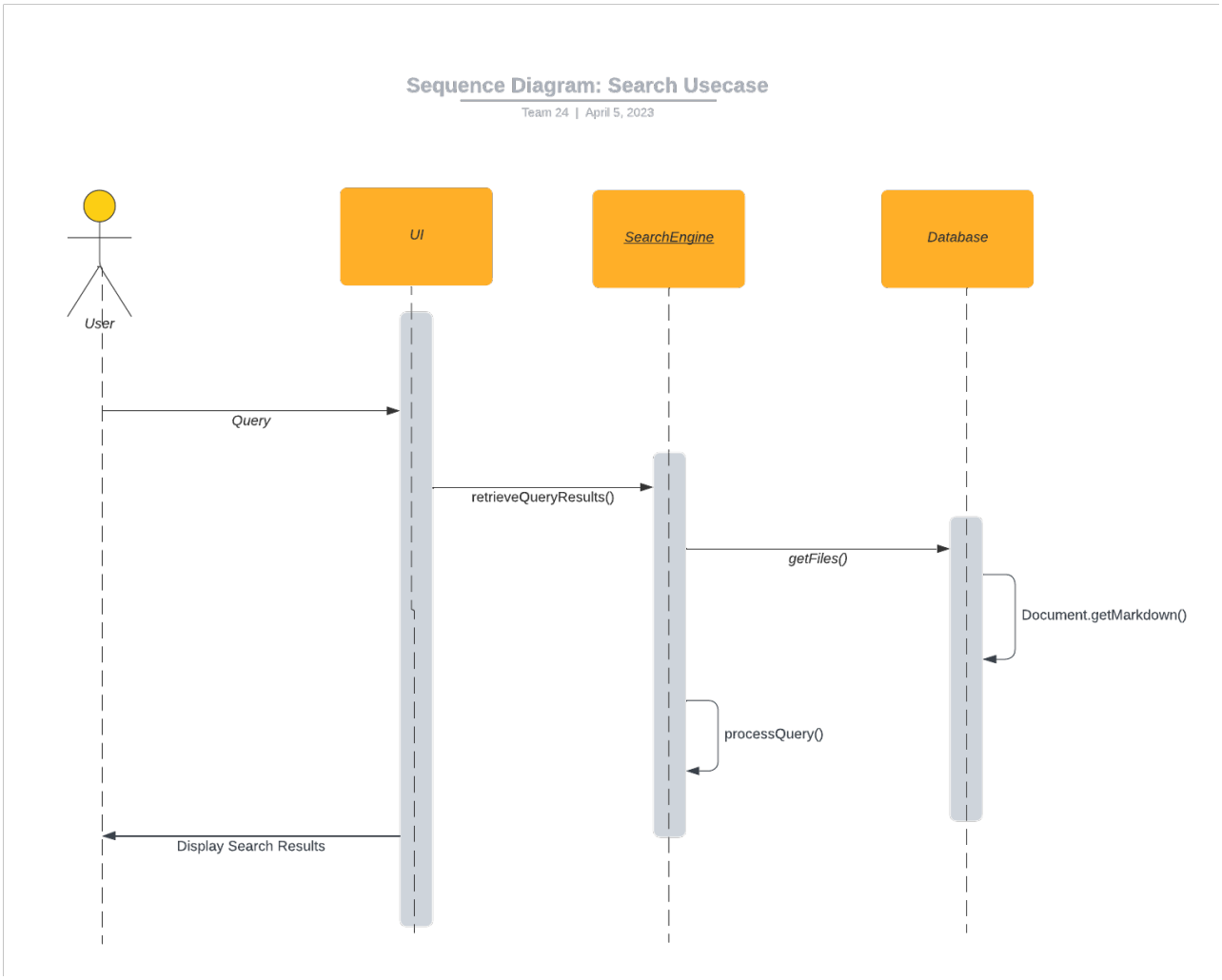

Figure 2: Sequence Diagram for Search use case

3.2 Use Case 2: Upload Files

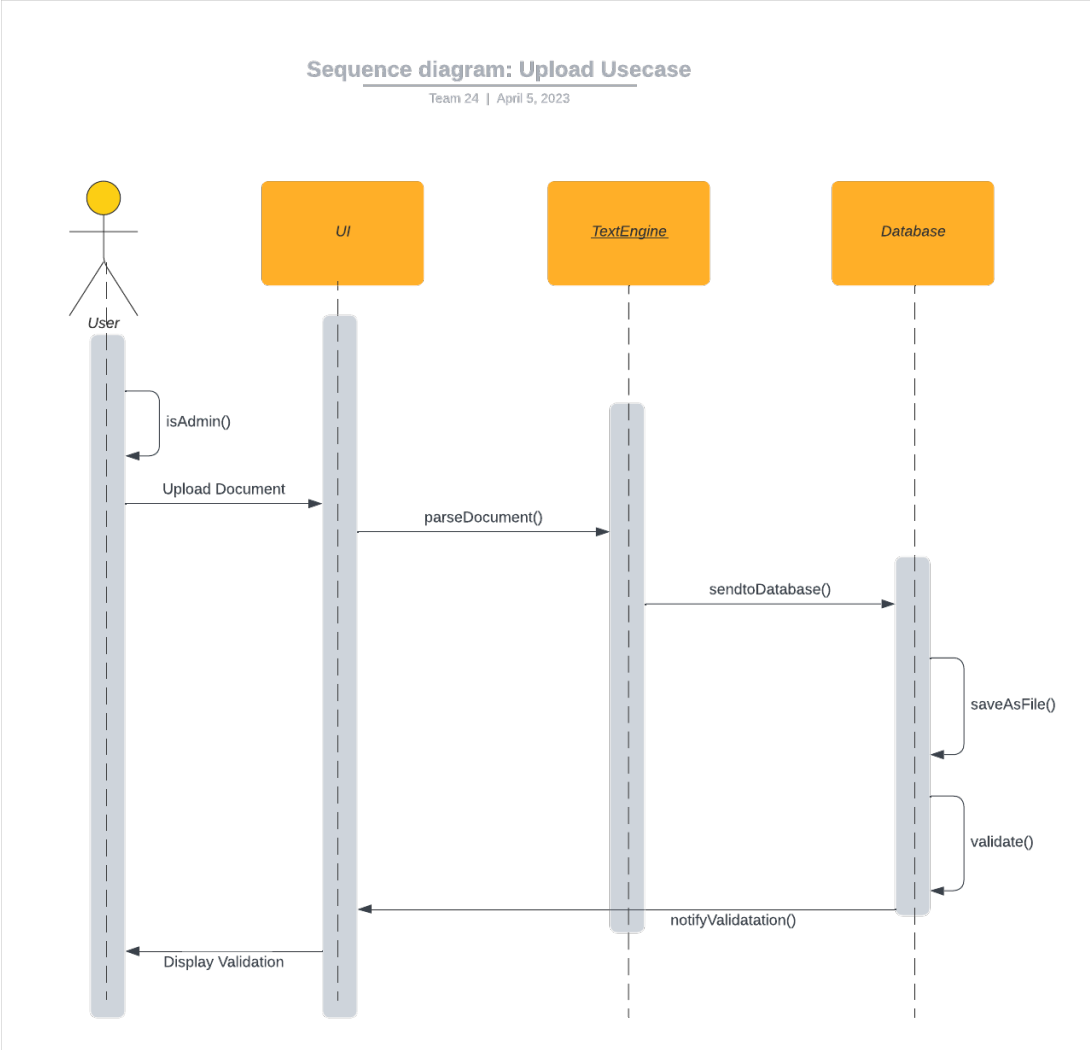

Figure 3: Sequence Diagram for Upload File use case

4 Design Patterns

Software design patterns are solutions to commonly occurring problems and scenarios in software design, hence the name. Typically, design patterns provide a general solution that can be adapted appropriately to fit the specific needs of a particular system. In the Minutes Management System, We have identified that the following design patterns are/will be integrated into our code. They are:

1. **Singleton Pattern:** The Singleton pattern is a design pattern that restricts the instantiation of a class to a singular instance, hence the name Singleton. There is easy access to this object, which acts as a global entity.

In the Minutes Management System, such a pattern is ideal for the TextEngine and SearchEngine classes. A singular parser is required to parse all the different documents that are uploaded to the system; the parser processes the input documents and generates the content as raw text before storing it in the database with the appropriate metadata.

Similarly, a central Natural Language Processing (NLP) model (possibly pretrained) will process the user search queries before retrieving the relevant results from our existing corpus of documents; in other words, it is a text retrieval engine. Once again, only a singular global instantiation is required, which once again makes the singleton pattern an ideal pattern.

2. **Visitor Pattern:** The Visitor pattern is generally used when new functionalities are to be introduced without tampering with the original class hierarchy.

   The Visitor pattern can be extremely helpful in a project such as this, where it might be necessary to integrate new functionalities later down the line without affecting the class hierarchy. For example, there could be a need to introduce some new method to objects of the Document class, but the function might be overridden differently for the different subclasses such as Minutes, Agenda, and Handbook. In this case, it would be more convenient to incorporate the visitor pattern to introduce these new functionalities elegantly and cohesively as a separate collection without interfering with the existing class structure.

3. **Strategy Pattern:** The Strategy patterns allows the definition of a family of algorithms and their interchangeable usage.

   In the Minutes Management System, this can be applied to the context of parsing. The TextEngine class, which is responsible for parsing PDF/document text to our document format, shall be composed of different parsing strategies. These can then be used interchangeably depending on the various parameters of a document, and the type of document which needs to be parsed.

   Since some strategies might be common across various types, and will be general enough to not require specific document context, it is better to encapsulate these strategies in a different hierarchy. This will also allow us to efficiently improve/add new strategies in the future.