Backend Design

Logic View

The project will use Spring Boot Java EE framework to develop. It has several benefits:

1. **Rapid Development**: Spring Boot simplifies and accelerates development with its opinionated defaults and auto-configuration, allowing us to focus on coding business logic.
2. **Microservices**: Spring Boot supports microservices architecture, enabling the development of modular, scalable, and independently deployable components.
3. **Reduced Boilerplate**: Spring Boot reduces boilerplate code and configuration, leading to concise and maintainable codebases.
4. **Integrated Ecosystem**: It integrates seamlessly with various Spring projects and libraries, providing a wide range of tools for different needs like data access, security, and more.
5. **Embeddable Servers**: Spring Boot apps can be packaged with embedded servers like Tomcat, Jetty, or Undertow, simplifying deployment and reducing external dependencies.
6. **Easy Testing**: Spring Boot offers testing support, making unit and integration testing straightforward, enhancing code quality and reliability.
7. **Cloud-Native**: Spring Boot supports cloud-native development and deployment, making it well-suited for modern cloud environments.
8. **Customizable**: While providing sensible defaults, Spring Boot remains highly customizable, allowing developers to tailor solutions to specific project requirements.

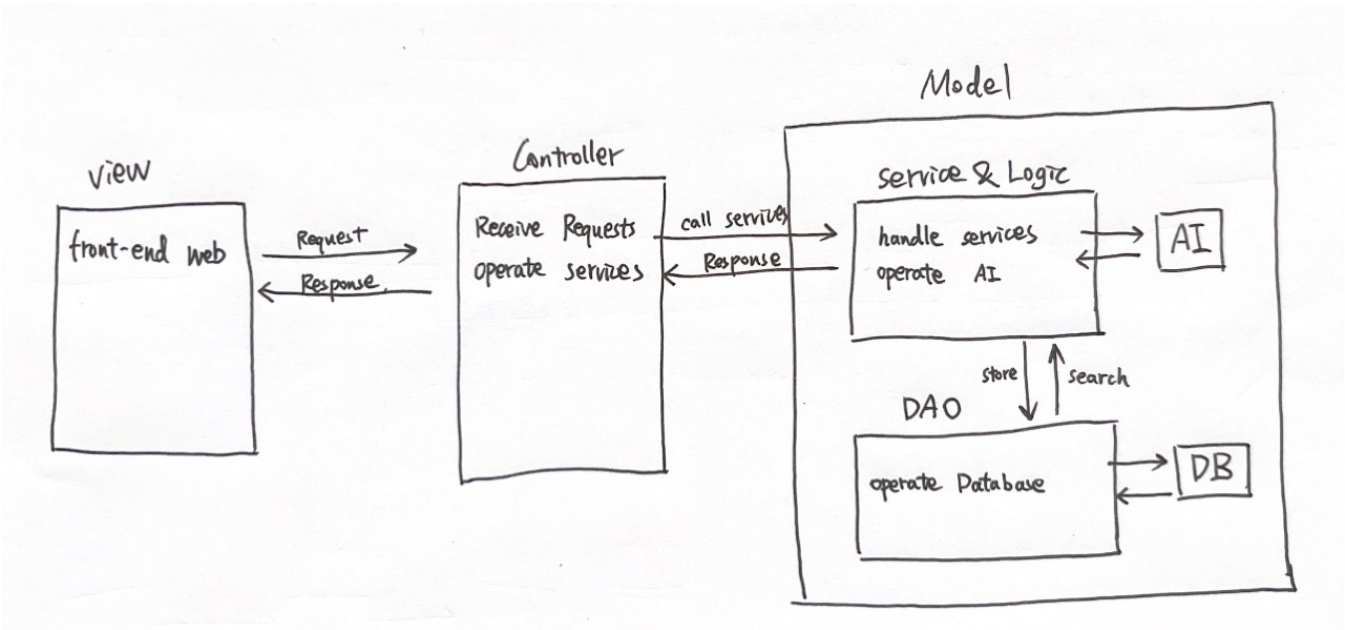
Besides, our team is going to apply a combination of MVC (Model-View-Controller) Architecture and Three Major Components for application design and development.

MVC is a software architectural pattern that separates an application into three interconnected components to manage and organize the codebase effectively.

- **Model**: Represents the application's data and business logic. It manages data storage, retrieval, and manipulation. The model component is responsible for maintaining the integrity and consistency of the data.
- **View**: Handles the presentation and user interface. It displays the data to the user and captures user interactions. Views are responsible for rendering data in a format that is understandable and visually appealing. This is the front-end section.
- **Controller**: Manages user input and acts as an intermediary between the model and the view. It processes user requests, updates the model accordingly, and communicates with the view to reflect changes.

As for the Three Major Components, there are three layers:

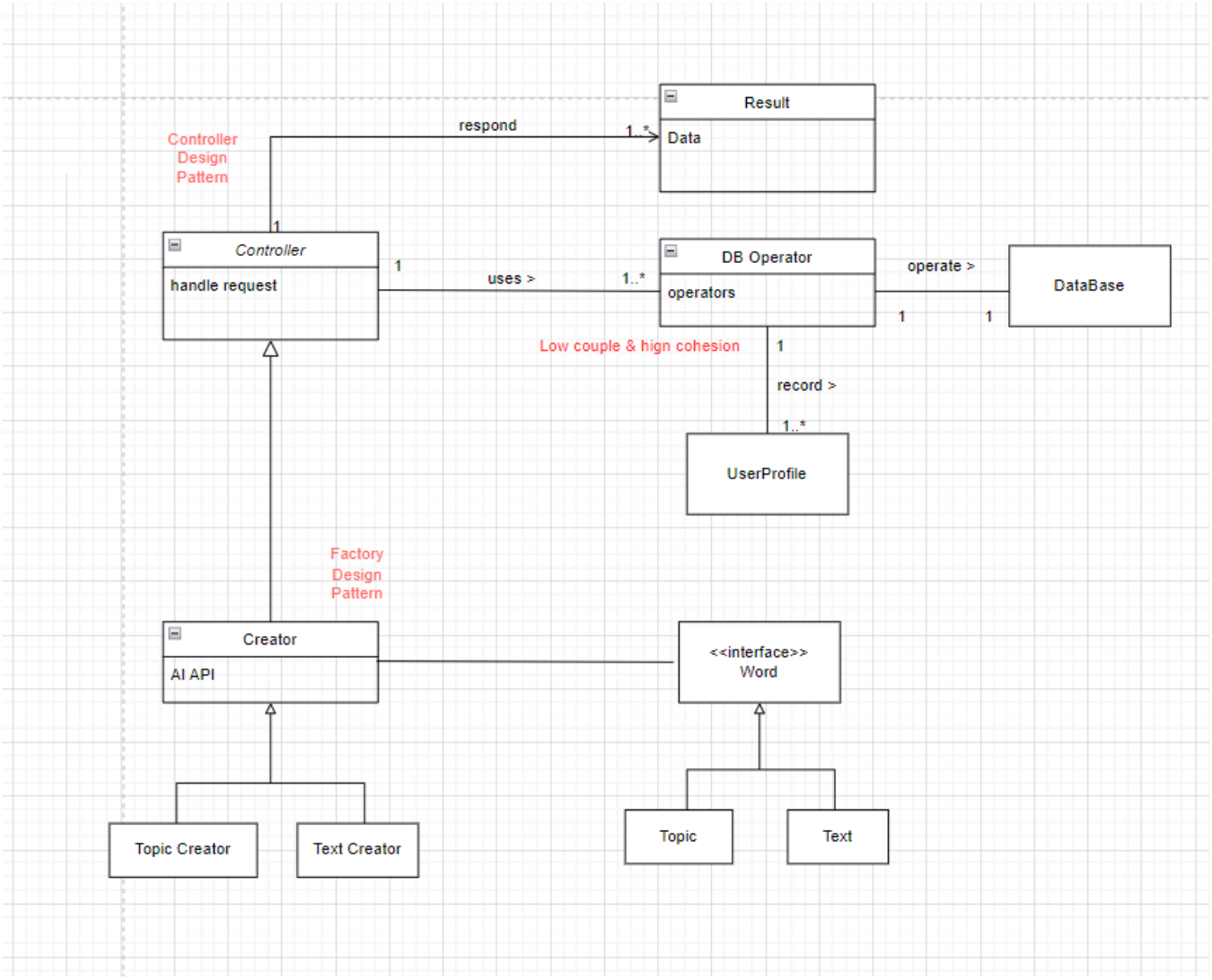
1. **Controller Layer**: The controller layer handles data from front-end and coordinates other layers to give expected response back. It is same as the Controller in the MVC architecture.
2. **Service Layer**: The service layer contains the business logic of the application. It acts as an intermediary between the controller and the data access layer (DAO). The service layer coordinates different actions and orchestrates interactions between multiple data entities to fulfill the requirements defined by the user.
3. **Data Access Layer (DAO)**: The DAO layer is responsible for interacting with the data storage and retrieval mechanisms, such as databases or external APIs. It abstracts the data access operations, providing an interface for the rest of the application to interact with the data storage without needing to know the underlying implementation details.



In our design, as shown in the diagram above, both service and DAO layers are in the Model component. The serive layer will interact with AI component for topic and text generation features of the application. The DAO layer will interact with data base for data storage and login system construction.

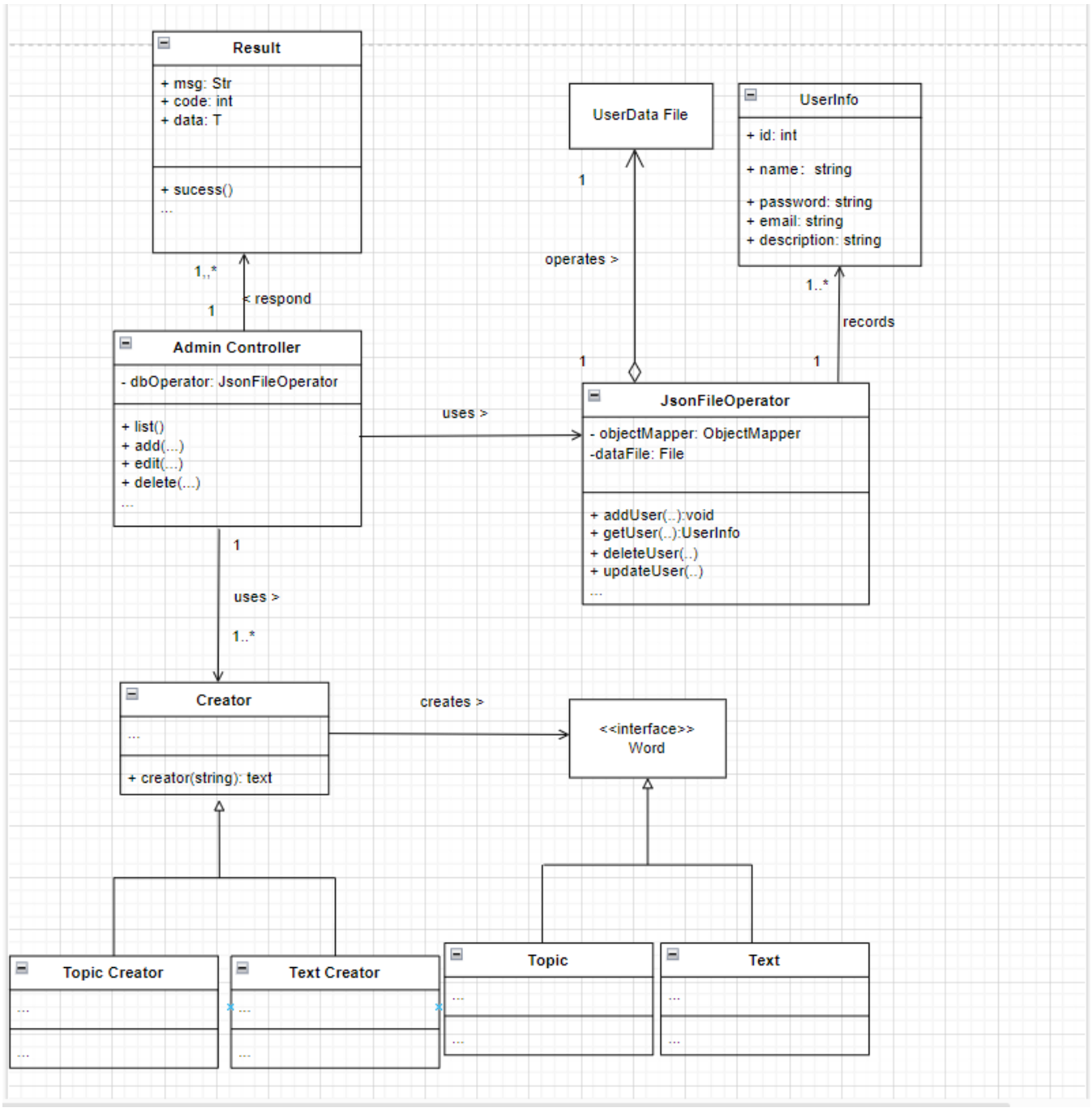
Domain Model

The domain model of the backend architecture design is below:



Design Class Model

The design class diagram is apllied for structure design, as shown below:



The architecture applies **factory pattern** which is a creational design pattern used in object-oriented programming. It provides an interface for creating objects in a super class, but allows subclasses to alter the type of objects that will be created. It is a way to encapsulate the object creation process and make it more flexible, extensible, and maintainable.

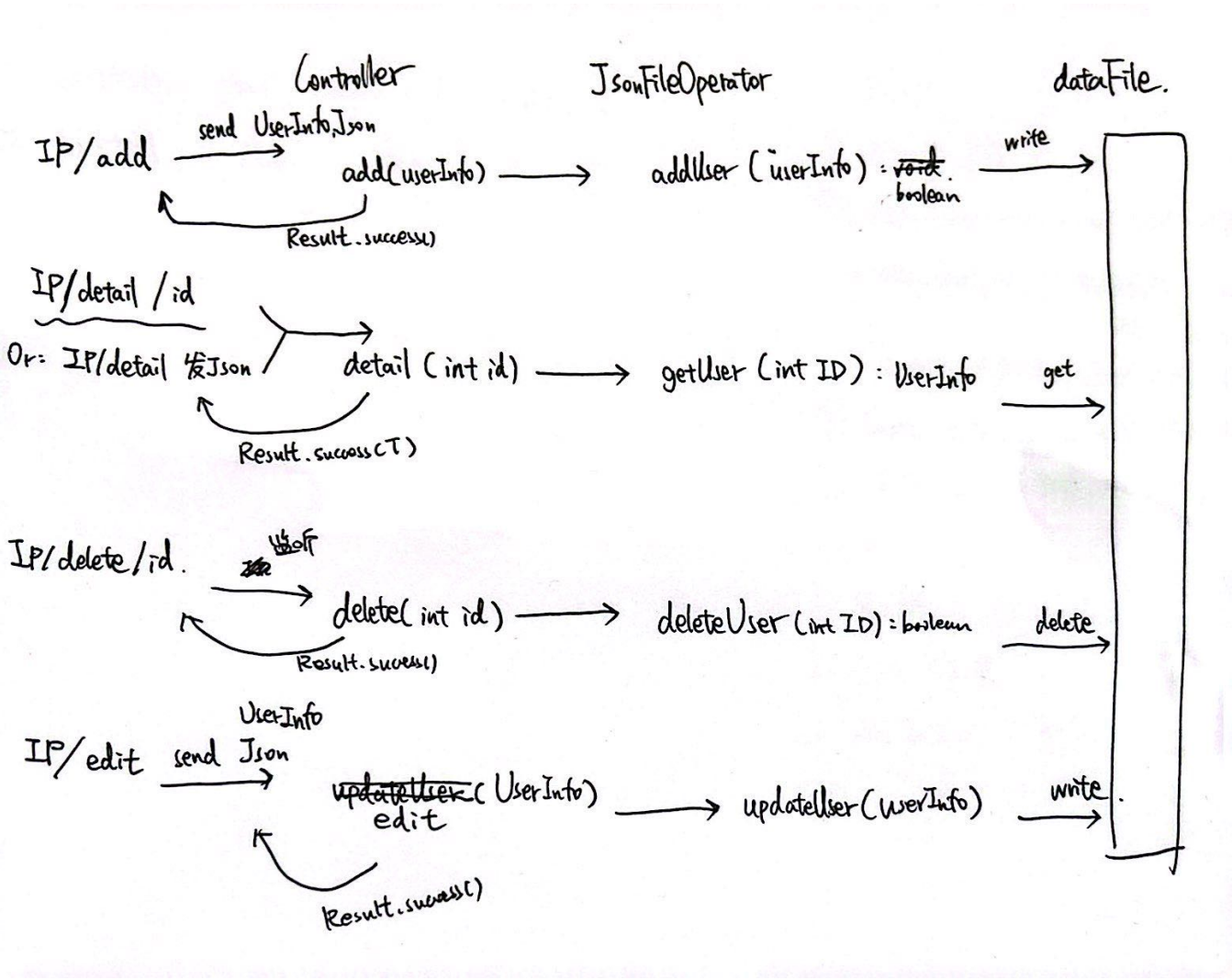
Key components of the Factory Design Pattern:

1. **Creator (Factory)**: This is an abstract class or interface that defines a method for creating objects. The Creator may also include some default implementation for common object creation. The Creator class above is the overal factory representation.
2. **Concrete Creator**: These are subclasses or implementations of the Creator interface. Each Concrete Creator provides its own implementation of the factory method to create specific types of objects. They are responsible for instantiating and returning concrete objects. Topic Creator and Text Creator are the conrete creators.
3. **Product**: This is an interface or abstract class that defines the interface for the objects created by the factory. The Product can have common attributes and methods that all concrete products must implement. In this case, the Word interface is the Product.
4. **Concrete Product**: These are the actual objects that are created by the Concrete Creators. Each Concrete Creator is associated with one or more Concrete Products. Topic and Text classes are the concrete product here.

With the JsonFileOperator, the system achieves relatively high cohesion and low coupling.

Process View

Currently Frontend and backend communication protocol design:



Development View

Requirement	what	Why	How
Connect DataBase	The system should be able to connect a Database for data storage.	To achieve user login system.	The system connects to a local Json file as file storage.
Add new user	The system should be able to add new user to the DB.	To achieve user registeration function.	Load the current Json file and write data on it.
Search user	The system should search user's information based on primary key.	To achieve user verification function,	Load the Json file, and use primary key as key to search the relevant value pair.
Edit current user	The system should be able to update a user's information.	To aviod user cannot change their profile.	Rewrite the user's value pair data on Json file.
Delete user	The system should be able to remove a user.	To achieve user removal function.	Clear the key value pair of that user on the file.
Confidentiality	The system should ensure the sensitive information is only accessible to those who are authenticated.	To protect sensitive data from being abused.	The system has admin user who has the access only.
Data persistence	The system should ensure the data is saved properly and can be accessed later.	To aviod data getting lost and causing problems for users.	Only the server has access to the data file and handle it correctly.
Extensibility	The system should be able to support entension and addition of new features in the future.	To allow further development.	System components have low coupling and high cohesion. All classes are reusable.

Concurrency	The system should be able to support multiple users using the system the same time.	To allow reliable user experience.	Choose to deploy on a service tht can implement concurrency well. i.e. Heroku