# Architecture Pattern For Journalify

A desktop based
Daily Journaling Application

**Course No:** CSE 3106
**Course Title:** Software Development Project

| Submitted to: | Submitted by: |
|---|---|
| *Dr. Amit Kumar Mondal*<br>Associate Professor<br>Computer Science and Engineering<br>Discipline,<br>Khulna University,Khulna | Name: Uma Datta<br>Student ID: **210226**<br>&<br>Name:Shaikh Jaberul Islam Moin<br>Student ID: **210238**<br>3 rd Year, 1 st Term,<br>Computer Science and Engineering<br>Discipline,<br>Khulna University,Khulna |

**Date of submission:**12.02.2024

## **Project Title:** Journalify,

A desktop-based Journaling Application.

## **The Architecture Pattern we determined for:**

Model-View-Controller(MVC) Pattern

To manage particular functions like we proposed: user authentication, mood tracking, and location tagging, customizable templates within the **Model** component, we wish to create distinct modules or classes. The **Controller** (in MVC) would manage the interaction logic and data flow between the View and the Model, while the **View** would be composed of graphical elements intended for user interaction. In general, as the MVC pattern supports concern separation, which facilitates the management and upkeep of intricate desktop applications. By maintaining the separation of presentation logic from the main logic of the application, it will improve testability, scalability, and code reuse.

# Let's look deeper into the Model-View-Controller (MVC) pattern in our project:

## Model:

**Description:**   The data and business logic of the application are represented by the Model component. It includes the operations for data manipulation, data validation, and data structure. It essentially controls the behavior and state of the application.

## Accountabilities:

1. Manages data persistence (such as saving journal entries, mood data, user authentication details, encryption details, location information, and custom templates).
2. Implements business logic (such as handling mood tracking algorithms, authentication processes,encryption algorithms, location tagging functionalities and customizable templates structure ).
3. Responds to requests from the Controller for data manipulation or retrieval.
4. Notifies the View of any changes in the data so that the interface can be updated accordingly (via the Observer pattern or similar mechanisms)

# View:

**Description:**   The View represents the presentation layer of the application. It is in the role of presenting data to the user in a readable manner and displaying the user interface. User input is received by the View, which then sends it to the Controller for processing.

## Accountabilities:

1. Renders graphical elements and user interface components (such as text fields, buttons, mood selection widgets, customizable templates and maps for location tagging).
2. Displays data retrieved from the Model to the user.
3. Records user input and sends it to the Controller for additional processing (e.g., journal entries, mood selections, custom interfaces and authentication credentials).
4. Can be updated to reflect modifications made to the application's state by either the Controller or the Model.

# Controller:

**Description:**    The Controller serves as the link between the View and the Model. It responds to requests, manages user input, and updates the Model or the View as necessary. The Controller manages data flow and sets the application's behavior in response to user input.

## Accountabilities:

- Receives user input from the View and translates it into actions to be performed by the Model.
- Carries out interactions with the Model to retrieve or modify data in response to user requests.
- Updates the View to give the user feedback or to reflect changes in the state of the Model.
- Implements application logic to control how data is processed and presented.
- Separates the user interface from the actual information and business logic, allowing for better modularity and maintainability.

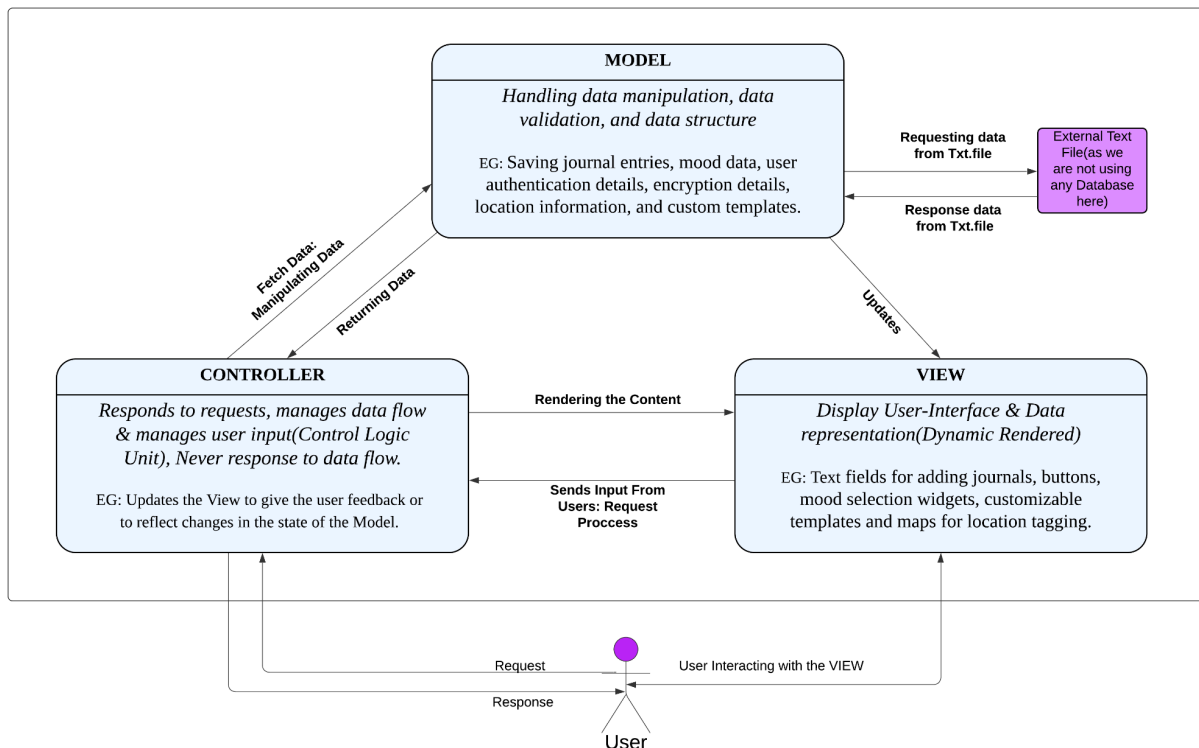## Graphical Representation of MVC pattern:



**MODEL**

*Handling data manipulation, data validation, and data structure*

EG: Saving journal entries, mood data, user authentication details, encryption details, location information, and custom templates.

Requesting data from Txt.file

External Text File(as we are not using any Database here)

Response data from Txt.file

Fetch Data: Manipulating Data

Returning Data

Updates

**CONTROLLER**

*Responds to requests, manages data flow & manages user input(Control Logic Unit), Never response to data flow.*

EG: Updates the View to give the user feedback or to reflect changes in the state of the Model.

Rendering the Content

Sends Input From Users: Request Proccess

**VIEW**

*Display User-Interface & Data representation(Dynamic Rendered)*

EG: Text fields for adding journals, buttons, mood selection widgets, customizable templates and maps for location tagging.

Request

User Interacting with the VIEW

Response

User

## Fig: Diagram of MVC in Journalify(Link of the Journalify diagram)