# Documentation

# Project Structure

WORKING\_WITH\_TABBED\_PANES

├── .idea

├── out

├── src

│ └── main

│ └── java

│ └── WorkingWithTabbedPanes

│ ├── Controller

│ │ ├── MainApp.java

│ │ ├── MainController.java

│ │ ├── MainFrame.java

│ ├── Model

│ │ ├── Doctors.java

│ │ ├── Patients.java

│ ├── Views

│ ├── HomePanel.java

│ ├── DoctorsPanel.java

│ ├── PatientsPanel.java

│ ├── ContactFormPanel.java // Nouveau panneau pour le formulaire de contact

│ ├── InfoPanel.java // Nouveau panneau pour afficher des informations

│ ├── TabbedPaneFrame.java // Nouveau cadre principal utilisant JTabbedPane

├── .gitignore

├── pom.xml

├── WORKING\_WITH\_TABBED\_PANES.iml

# 2.1 Technology Stack

Languages and Frameworks: The development of this project utilized Java as the primary programming language, specifically leveraging Java Swing for the graphical user interface (GUI) development. Java Swing provides a robust framework for building interactive and user-friendly desktop applications. Additionally, we incorporated the use of the Java Collections Framework to manage data structures efficiently.

Development Tools: The primary integrated development environment (IDE) used for this project was IntelliJ IDEA, chosen for its powerful features and ease of use. Version control was managed using Git, with repositories hosted on GitHub to facilitate collaboration and code management. ~~Continuous Integration/Continuous Deployment (CI/CD) was achieved through GitHub Actions, enabling automated testing and deployment workflows.~~

# 2.2 System Architecture

High-Level Design: The system architecture follows a Model-View-Controller (MVC) pattern to separate concerns and enhance maintainability. The Model represents the data and business logic, the View is responsible for the UI components, and the Controller manages the communication between the Model and the View.

~~Modules: The system is divided into several key modules. The Model contains classes representing the application's data, such as Doctors and Patients. The View consists of UI components, including panels for displaying and interacting with data. The Controller handles the logic and coordination between the Model and the View.~~

# 2.3 Features and Functional Workflow

Key Features: The application includes several key features. The tabbed interface provides a user-friendly interface with multiple tabs for different sections. CRUD operations allow users to create, read, update, and delete doctor and patient information. The contact form provides a means for users to submit their contact information and messages. An information panel displays instructions and important messages for users.

User Workflow: Users start at the HomePanel with a welcome message. They navigate between tabs to access Doctors, Patients, Contact, and Info sections. Within the Doctors and Patients panels, users can add, edit, delete, and refresh data. Users fill out and submit the contact form in the Contact section. Important information and instructions are displayed in the InfoPanel.

# 2.4 Development Process

Implementation Details: Core functionalities were implemented using Java Swing components. Data tables were created using `JTable` and `DefaultTableModel`, while forms utilized `JTextField` and `JTextArea` components. Action listeners were added to buttons to handle user interactions.

Code Organization: The project directory is structured as follows. The `src/main/java` directory contains all Java source files. The `Controller` package includes the main application entry point and controller classes. The `Model` package contains data model classes. The `Views` package houses all UI components.

Third-party Libraries/Integrations: No external libraries were used in this project, ensuring a lightweight and dependency-free application.

# 2.5 Interface Design

UI Elements: The interface design focused on simplicity and usability. Each panel is designed to provide clear and concise interaction options. Key UI elements include tabbed panes for navigation, data tables for displaying information, and input forms for data entry.

## Screenshots:

Key interface elements with annotations will be added here in the Word document.

# 2.6 Challenges and Resolutions

Development Challenges: The project faced several development challenges. Synchronization issues arose when ensuring data consistency across different panels. Managing complex layouts and ensuring responsiveness across different screen sizes also proved challenging.

Solutions Implemented: Data binding was implemented between models and views to ensure synchronization. Java Swing's layout managers, such as `GridBagLayout`, were utilized to create flexible and adaptive UI layouts.

# 2.7 Future Enhancements

Potential Improvements: Future improvements could include adding search functionality to quickly locate doctors and patients. Implementing features to generate and export reports based on the data would also enhance the application.

Scalability Plans: To improve scalability, transitioning to a database-backed system to manage larger datasets is recommended. Further breaking down the system into more modular components would facilitate easier maintenance and scalability.

# Class Description

# Class: MainApp

## Description:

* MainApp is the entry point of the application.
* It initializes the MainFrame with Tabbed Panes.

## Usage:

java src.main.java.WorkingWithTabbedPanes.Controller.MainApp

## Details:

* This class runs the application on the Event Dispatch Thread (EDT) to ensure thread safety when creating and manipulating Swing components.
* The MainFrame class is initialized within the `run()` method of the `SwingUtilities.invokeLater` method.

## Example:

SwingUtilities.invokeLater(() -> {

try {

new MainFrame();

} catch (Exception e) {

e.printStackTrace();

}

});

## References:

* `javax.swing.SwingUtilities`
* `src.main.java.WorkingWithTabbedPanes.Controller.MainFrame`

Class MainFrame

1. Description

* MainFrame is the main frame of the application that contains the Tabbed Panes.
* It provides a graphical user interface (GUI) with multiple tabs for different sections of the application.

## Details:

* The class creates a `JTabbedPane` with five tabs: Home, Doctors, Patients, Contact, and Info.
* Each tab contains a different panel corresponding to its functionality.

## Tabs:

1. Home: Displays the HomePanel.
2. Doctors: Displays the DoctorsPanel.
3. Patients: Displays the PatientsPanel.
4. Contact: Displays the ContactFormPanel.
5. Info: Displays the InfoPanel.

## Usage:

SwingUtilities.invokeLater(() -> new MainFrame());

## References:

* `javax.swing.JFrame`
* `javax.swing.JTabbedPane`
* `src.main.java.WorkingWithTabbedPanes.Views.HomePanel`
* `src.main.java.WorkingWithTabbedPanes.Views.DoctorsPanel`
* `src.main.java.WorkingWithTabbedPanes.Views.PatientsPanel`
* `src.main.java.WorkingWithTabbedPanes.Views.ContactFormPanel`
* `src.main.java.WorkingWithTabbedPanes.Views.InfoPanel`

# Class HomePanel

## Description:

* HomePanel is the welcome screen of the application.
* It displays a welcome message and an enter button to navigate to the menu.

## Details:

* The panel uses a `GridBagLayout` for flexible component arrangement.
* It includes an icon, a welcome message, and an enter button.

# Components:

1. Icon: Displays an image icon.
2. Welcome Label: Displays a welcome message.
3. Enter Button: When clicked, it displays a dialog message.

## **Usage**:

HomePanel homePanel = new HomePanel();

homePanel.setVisible(true);

## 5. References:

* `javax.swing.JPanel`
* `javax.swing.JLabel`
* `javax.swing.JButton`
* `javax.swing.ImageIcon`
* `java.awt.GridBagLayout`
* `java.awt.GridBagConstraints`

# Class DoctorsPanel

DoctorsPanel is a panel that displays a list of doctors with their details. It uses a DefaultTableModel to display data in a table and provides buttons to add, edit, delete, and refresh doctor information. The panel includes a menu bar for navigation and data management, and uses methods to configure components, handle button actions, and update displayed data.

# Class PatientsPanel

PatientsPanel is a panel that displays a list of patients with their details. It uses a DefaultTableModel to display data in a table and provides buttons to add, edit, delete, and refresh patient information. The panel includes a menu bar for navigation and data management, and uses methods to configure components, handle button actions, and update displayed data. It employs a GridBagLayout for arranging components and allows users to interact with the data using dialog boxes for input and confirmation.

# Class ContactFormPanel

ContactFormPanel is a panel that provides a contact form for users to submit their information and messages. It uses a GridBagLayout to arrange components such as text fields for the user's name and email, a text area for the message, and a submit button. When the submit button is clicked, the entered information is displayed in a dialog box. The panel is designed to collect user input in a structured manner and provide feedback upon submission.

# Class InfoPanel

InfoPanel is a panel that displays static or dynamically generated information such as instructions, messages, or announcements. It uses a JTextArea wrapped in a JScrollPane to ensure that the content can be scrolled if it exceeds the visible area. The text area is set to be non-editable and uses word wrapping to format the text neatly. This panel is designed to provide users with important information and instructions related to the application.

# Class Doctors

Doctors is a model class representing a doctor with their attributes. It includes details such as ID, name, and specialization. The class provides getter and setter methods for each attribute and overrides the toString method to display doctor details in a readable format. This class is essential for managing doctor data within the application and ensuring that doctor information is properly encapsulated and accessible.

# Class Patients

Patients is a model class representing a patient with their attributes. It includes details such as ID, name, age, and diagnosis. The class provides getter and setter methods for each attribute and overrides the toString method to display patient details in a readable format. This class is essential for managing patient data within the application and ensuring that patient information is properly encapsulated and accessible.

# Class MainController

MainController handles the navigation between different views and manages data interactions. It maintains lists of doctors and patients and provides methods to retrieve these lists. The class loads sample data for demonstration purposes and includes methods to interact with the data. This class is essential for managing the core logic of the application and ensuring that data is properly handled and accessible to different panels.

====================================================================