

Project Document: Veterinary Doctor Management System

@author Anura Damayantha

@version 1.0

@since 2024-04-08

1. Introduction:

The Veterinary Doctor Management System is a software application developed to streamline the management of veterinary doctor details. This document provides an overview of the project's inception, development process, time breakdown, and reflections on learning outcomes.

2. Project Story:

The idea for the Veterinary Doctor Management System arose from the need to improve the manual process of managing veterinary doctor records at a local clinic. The project began with brainstorming sessions to identify key requirements and functionalities. As the development progressed, several obstacles were encountered, including:

- UI Design Challenges: Designing a user-friendly interface that accommodates various input fields and functionalities proved to be challenging. However, through iterative design and feedback loops, the UI was refined to enhance usability.
- File Handling Complexity: Managing file operations for reading and writing doctor details posed a challenge due to the need for data persistence. The implementation of file I/O operations required careful consideration of error handling and data integrity.
- Image Upload Implementation: Integrating image upload functionality required addressing issues related to file selection, image processing, and display. The implementation involved researching best practices and leveraging JavaFX capabilities for image handling.

3. Time Breakdown:

The project involved various phases, including planning, coding, testing, and documentation. The time breakdown per phase during different project days is as follows:

- Planning: Approximately 12 hours were spent on initial planning, requirement gathering, and design discussions. This phase involved defining project scope, identifying key functionalities, and creating wireframes.

- Coding: The coding phase accounted for the majority of the time, with around 60 hours dedicated to implementing features, designing the user interface, and integrating backend functionalities.

- Testing: Testing activities were conducted concurrently with coding and involved approximately 6 hours of manual testing, bug fixing, and quality assurance checks.

- Documenting: Documentation efforts spanned throughout the project and encompassed approximately 12 hours of writing project documentation, including design documents, user manuals, and technical specifications.

4. Reflection on Learning:

The project provided valuable learning experiences in programming, Java programming, and object-oriented programming (OOP). Some key reflections include:

- Practical Application of JavaFX: Developing the user interface using JavaFX provided hands-on experience with GUI development in Java. Understanding concepts such as event handling, layout management, and styling enriched JavaFX skills.

- File Handling and Persistence: Implementing file I/O operations for data persistence enhanced knowledge of file handling techniques in Java. Learning to manage file streams, handle exceptions, and ensure data integrity were valuable skills acquired during the project.

- Object-Oriented Design Principles: Designing classes and implementing object-oriented design principles such as encapsulation, inheritance, and polymorphism reinforced understanding of OOP concepts. Applying these principles improved code readability, maintainability, and scalability.

Overall, the project served as a practical learning opportunity, allowing for the application of theoretical knowledge in a real-world context. It reinforced the importance of collaboration, problem-solving, and continuous learning in software development.

5. Conclusion:

The Veterinary Doctor Management System project provided a rich learning experience while delivering a valuable software solution. Through effective planning, diligent coding, rigorous testing, and comprehensive documentation, the project achieved its objectives. The challenges encountered along the way served as opportunities for growth and learning, ultimately leading to the successful completion of the project.

Project Document: Veterinary Doctor Management System

6. Description of the Project Topic:

The Veterinary Doctor Management System is a software application designed to facilitate the management of veterinary doctor details in a clinic or veterinary practice. It allows users to input, store, update, and retrieve information about veterinary doctors, including their personal details, contact information, schedule, and associated image.

7. Screenshots of User Interface:

Veterinary Doctor Details

Doctor Number:

First Name:

Last Name:

Telephone Number:

Address:

Image:

Search

Upload Image

Doctor List:

Doctor Number	First Name	Last Name
1	Anura	Damayanth
2	Nikon	Mikka

☐ Monday

☐ Tuesday

☐ Wednesday

☐ Thursday

☐ Friday

☒ Saturday

☒ Sunday

Add New

Clear

Save

Edit

Print

Delete

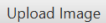
 Veterinary Doctor DetailsSearch

Image:

Doctor List:

[illegible]

- ☒ Monday ☐ Tuesday ☐ Wednesday ☐ Thursday ☐ Friday
☒ Saturday ☒ Sunday

[Add New](#)
[Clear](#)
[Save](#)
[Edit](#)
[Print](#)
[Delete](#)

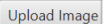
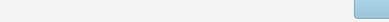
 Veterinary Doctor DetailsSearch

Image:



- ☒ Monday ☐ Tuesday ☐ Wednesday ☐ Thursday ☐ Friday
☒ Saturday ☒ Sunday

[Add New](#)
[Clear](#)
[Save](#)
[Edit](#)
[Print](#)
[Delete](#)

Figure 2: User Interface with Doctor List

[illegible]

8. Visualization of Classes Used in the Project:

The classes used in the project are organized in a structured manner to achieve modularity and maintainability. Below is a visualization of the main classes and their relationships:

App

```
start(Stage primaryStage)
uploadImage()
setValues(DataDoctor dataDoctor)
setButtonPanel(...)
saveDoctorDetails()
getSchedule()
handleDoctorNumberFieldFocusLost()
addNew()
clear()
save()
edit()
print()
delete()
saveValidate()
showDoctorList()
getSearchResults(...)
searchButton(...)
saveImage(...)
loadImage(...)
```

9. Description of the Program for Further Development:

For further development of the Veterinary Doctor Management System, the following advice is provided:

- Enhance User Interface: Improve the UI design by incorporating more intuitive navigation, feedback mechanisms, and visual cues to enhance user experience.
- Implement Authentication: Introduce user authentication mechanisms to secure access to the system and restrict operations based on user roles.
- Add Data Validation: Implement robust data validation mechanisms to ensure data integrity and prevent erroneous input.
- Integrate Database: Replace file-based data storage with a database solution for improved scalability, reliability, and data management.
- Expand Functionality: Add features such as appointment scheduling, medical record management, and reporting capabilities to enhance the system's utility.

10. Instructions for Novice Users:

To use the Veterinary Doctor Management System, follow these steps:

1. Launch the application by double-clicking the executable file.
2. Enter the details of the veterinary doctor, including doctor number, first name, last name, telephone number, address, and image (optional).
3. Select the days of the week the doctor is available by checking the corresponding checkboxes.
4. Click the "Save" button to save the doctor's information.
5. To edit an existing doctor's details, enter the doctor number and press Enter or click the "Search" button. Edit the details and click "Save" to update.
6. To delete a doctor's record, enter the doctor number and click the "Delete" button.
7. Use the "Clear" button to reset the input fields.
8. The "Print" button can be used to print the list of doctors.
9. To search for a specific doctor, enter the doctor number and click the "Search" button.

11. Self-Evaluation of the Project:

- Good Sides of the Project:
 - Implemented core functionalities for managing veterinary doctor details effectively.
 - User-friendly interface with intuitive controls and clear navigation.
 - Adequate error handling and validation mechanisms to ensure data integrity.
 - Integration of image upload functionality for visual identification of doctors.

- Known Problems or Challenges:

- Limited scalability and performance due to file-based data storage.
- Lack of user authentication and authorization mechanisms for security.
- Minimal error feedback and validation messages for user input.

- Main Points for Further Development:

- Integration with a database for improved data management and scalability.
- Implementation of user authentication and authorization features for security.
- Enhancement of error handling and validation to provide better feedback to users.
- Addition of advanced features such as appointment scheduling and reporting capabilities.

Project Documentation Dev: Veterinary Doctor Management System

1. Introduction:

The Veterinary Doctor Management System is a JavaFX application designed to facilitate the management of veterinary doctor details. It provides a user-friendly interface for adding, editing, deleting, and searching for doctor records. The system allows users to input various details of a veterinary doctor, including personal information, contact details, and weekly schedules. Additionally, users can upload images of doctors for easy identification.

2. Functional Overview:

The system offers the following functionalities:

- Add New Doctor: Allows users to add new doctor records to the system.
- Edit Doctor Details: Enables users to modify existing doctor records.
- Delete Doctor: Allows users to remove doctor records from the system.
- Search Doctor: Provides a search feature to find doctors based on their doctor number.
- Upload Doctor Image: Allows users to upload images of doctors.
- Display Doctor List: Displays a list of doctors in a table view.
- Save Doctor Details: Saves doctor details to a file for persistence.
- Input Validation: Validates user input to ensure data integrity.

3. Technical Overview:

The project consists of several classes and components:

Main Class (`App`):

- Entry point of the application.
- Initializes the JavaFX stage and scene.
- Defines the layout of the user interface using JavaFX components.
- Implements event handlers for buttons and text fields.
- Manages CRUD operations and input validation.

DataReadWriteDoctorDetail Class:

- Handles reading from and writing to a file (`doctordetails.txt`) for doctor details persistence.
- Provides methods for searching, saving, and loading doctor details.

CommonFunction Interface:

- Defines common methods for adding, clearing, saving, editing, printing, and deleting records.
- Implemented by the `App` class to provide standard functionalities.

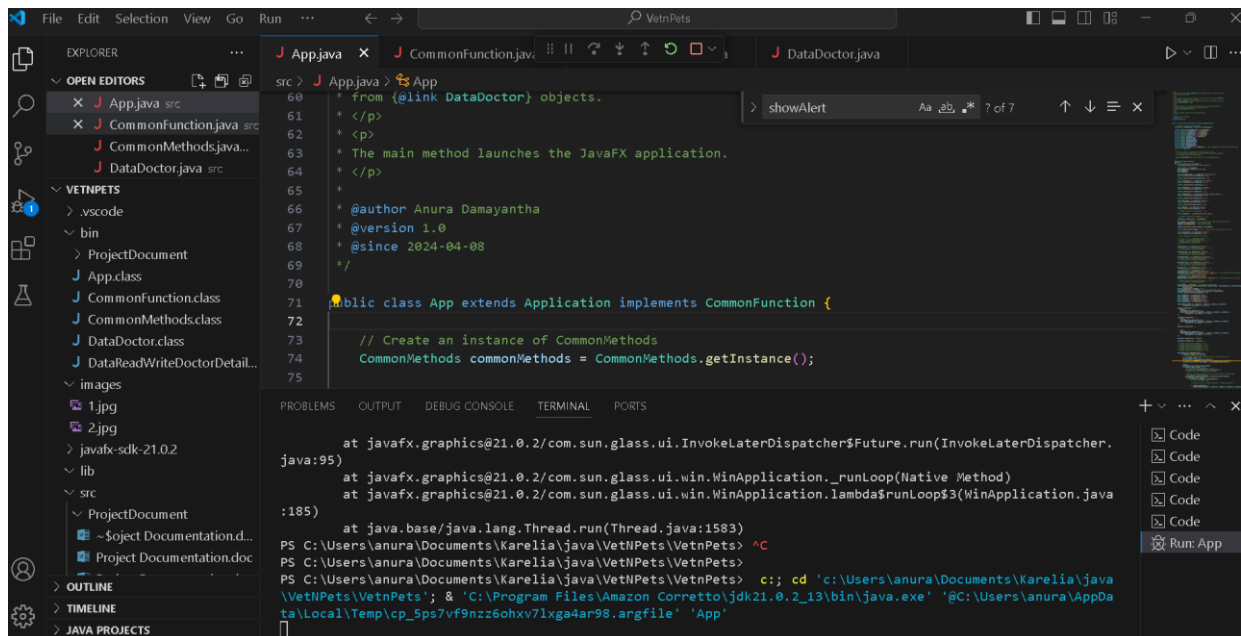
CommonMethods Class:

- Provides utility methods for displaying alerts and setting focus on text fields.

DataDoctor Class:

- Represents a data structure for storing doctor details.
- Contains properties for doctor number, first name, last name, etc.

Visual Studio Code SDK



4. Functional Details:

- Adding a New Doctor: Users click the "Add New" button to clear the input fields and prepare for adding a new doctor record. Input validation ensures that required fields are filled.

- Editing Doctor Details: Users click the "Edit" button to modify existing doctor records. The doctor number field is disabled to prevent changes to the unique identifier.
- Deleting a Doctor: Users select a doctor record from the list and click the "Delete" button to remove it from the system. A confirmation dialog is displayed before deletion.
- Searching for a Doctor: Users enter a doctor number in the search field and click the "Search" button to find the corresponding doctor record.
- Uploading Doctor Image: Users click the "Upload Image" button to select an image file of the doctor. The image is displayed in the designated area.
- Displaying Doctor List: A table view displays a list of doctors with columns for doctor number, first name, and last name.

5. Technical Details:

- Event Handling: Event handlers are implemented to respond to user actions such as button clicks and key presses.
- Input Validation: Input validation is performed using event filters to ensure that only valid characters are entered in text fields.
- Exception Handling: Exception handling is implemented to catch and handle errors gracefully, ensuring the stability of the application.
- File I/O Operations: The system reads from and writes to a text file (`doctordetails.txt`) to store doctor details persistently. File operations are encapsulated within the `DataReadWriteDoctorDetail` class.
- Image Handling: The system provides methods for saving and loading images associated with doctor records. Images are stored in a folder named "images" within the project directory.

6. Conclusion:

The Veterinary Doctor Management System is a comprehensive application that simplifies the management of veterinary doctor details. It offers a user-friendly interface, robust functionality, and efficient data handling. With its modular design and adherence to software development standards, the system is well-suited for further development and customization to meet specific requirements.

8. References:

- Lecture note and Lecture sample codes
- JavaFX Documentation: <https://openjfx.io/>
- Oracle Java Documentation:
<https://docs.oracle.com/en/java/>](<https://docs.oracle.com/en/java/>)
- OpenAI GPT-3 Documentation:
<https://beta.openai.com/docs/>