

UNIVERSIDAD VERACRUZANA FACULTAD DE INFORMATICA, XALAPA

EXPERIENCIA EDUCATIVA PROGRAMACION AVANZADA

PROYECTO

ALUMNOS:

Espinoza Rodríguez José Antonio Mattaí Martines Montero

ACADEMICO:

Lopez Herrera Juan Luis

FECHA DE ENTREGA: 13/06/2023

XALAPA-ENRÍQUEZ., VERACRUZ.

Main:

- 1. El código se encuentra en el paquete com.codetreatise.
- 2. El archivo contiene la definición de una clase llamada Main que extiende la clase Application de JavaFX.
- 3. La clase Main anotada con @SpringBootApplication indica que es una clase de aplicación Spring Boot.
- 4. La clase Main crea y administra el contexto de la aplicación Spring Boot.
- 5. En el método main, se inicia la aplicación JavaFX llamando al método launch() de la clase Application.
- 6. El método init() inicializa el contexto de Spring Boot.
- 7. El método start() se ejecuta al iniciar la aplicación y muestra una escena inicial a través del StageManager.
- 8. El método stop() se ejecuta al finalizar la aplicación y cierra el contexto de Spring Boot.
- 9. El método displayInitialScene() cambia a la escena de inicio de sesión.
- 10. El método springBootApplicationContext() crea un constructor de aplicaciones Spring Boot con la clase Main y devuelve el contexto de la aplicación.

Pet.java:

- 1. La clase está anotada con @Entity, lo que indica que se trata de una entidad que se puede almacenar en una base de datos.
- 2. La anotación @Table(name="Pet") especifica el nombre de la tabla en la base de datos donde se almacenarán los registros de las mascotas.
- 3. La clase tiene un conjunto de atributos que representan diferentes propiedades de una mascota, como id, firstName, lastName, dob (fecha de nacimiento), gender (género), role (rol), email y password.
- 4. El atributo id está anotado con @Id y @GeneratedValue(strategy=GenerationType.IDENTITY), lo que indica que es una clave primaria generada automáticamente.
- 5. Hay varios métodos de acceso (getters y setters) para cada atributo que permiten obtener y establecer sus valores.
- 6. El método toString() se sobrescribe para proporcionar una representación de cadena de la instancia de la clase Pet.

User.java:

- 1. La clase está anotada con @Entity, lo que indica que se trata de una entidad que se puede almacenar en una base de datos.
- 2. La anotación @Table(name="User") especifica el nombre de la tabla en la base de datos donde se almacenarán los registros de los usuarios.
- 3. La clase tiene un conjunto de atributos que representan diferentes propiedades de un usuario, como id, firstName, lastName, dob (fecha de nacimiento), gender (género), role (rol), email y password.
- 4. El atributo id está anotado con @Id y @GeneratedValue(strategy=GenerationType.IDENTITY), lo que indica que es una clave primaria generada automáticamente.

- 5. Hay varios métodos de acceso (getters y setters) para cada atributo que permiten obtener y establecer sus valores.
- 6. El método toString() se sobrescribe para proporcionar una representación de cadena de la instancia de la clase User.

AppJavaconfig:

- 1. La clase está anotada con @Configuration, lo que indica que es una clase de configuración de Spring.
- 2. La clase tiene métodos anotados con @Bean, los cuales son métodos de fábrica para la creación de diferentes beans en el contexto de Spring.
- 3. El método exceptionWriter() crea un bean de tipo ExceptionWriter que se utiliza para manejar excepciones.
- 4. El método resourceBundle() crea un bean de tipo ResourceBundle que se utiliza para cargar archivos de recursos.
- 5. El método stageManager() crea un bean de tipo StageManager que se utiliza para gestionar los escenarios en JavaFX.
- 6. Algunos métodos tienen anotaciones adicionales como @Scope("prototype") y @Lazy(value = true) para especificar el alcance y la carga diferida de los beans.

Spring:

- 1. La clase está anotada con @Component, lo que indica que es un componente gestionado por Spring.
- 2. La clase tiene dos campos finales: resourceBundle y context.
- 3. resourceBundle es un objeto ResourceBundle que se utiliza para cargar archivos de recursos, como archivos de idioma.
- 4. context es un objeto ApplicationContext de Spring que proporciona acceso a los beans y componentes del contexto de Spring.
- 5. La clase tiene un constructor que recibe el ApplicationContext y el ResourceBundle como argumentos, y los inicializa en los campos correspondientes.
- 6. La clase tiene un método load() que recibe la ruta de un archivo FXML como argumento y devuelve un objeto Parent que representa la jerarquía del nodo raíz del archivo FXML.
- 7. Dentro del método load(), se crea un objeto FXMLLoader que se utiliza para cargar y procesar el archivo FXML.
- 8. El FXMLLoader se configura para utilizar el ApplicationContext como factoría de controladores, lo que permite que los controladores FXML sean administrados por Spring.
- 9. También se configura el FXMLLoader con el ResourceBundle para proporcionar soporte de internacionalización.
- 10. Finalmente, se establece la ubicación del archivo FXML y se carga utilizando el método load() del FXMLLoader, y se devuelve el resultado.

StageManager:

1. La clase StageManager es una clase que se encarga de administrar la ventana principal (Stage) de una aplicación JavaFX. Aquí está la información clave sobre la clase:

- 2. La clase tiene un campo primaryStage de tipo Stage y un campo springFXMLLoader de tipo SpringFXMLLoader.
- 3. La clase tiene un constructor que recibe un objeto SpringFXMLLoader y un objeto Stage como argumentos, y los inicializa en los campos correspondientes.
- 4. La clase tiene un método switchScene() que recibe un objeto FxmlView como argumento. Este método carga la jerarquía de nodos de la vista asociada al FxmlView, y luego muestra la vista en la ventana principal.
- 5. El método show() es un método privado que recibe un nodo raíz (Parent) y un título (String). Este método prepara la escena y configura la ventana principal con la escena y el título proporcionados. Luego muestra la ventana principal.
- 6. El método prepareScene() es un método privado que recibe un nodo raíz (Parent) y prepara la escena. Si la escena aún no está configurada en la ventana principal, se crea una nueva escena y se establece como escena raíz.
- 7. El método loadViewNodeHierarchy() es un método privado que recibe la ruta de un archivo FXML y carga la jerarquía de nodos asociada al archivo FXML utilizando el SpringFXMLLoader.
- 8. El método logAndExit() es un método privado que recibe un mensaje de error (String) y una excepción (Exception). Este método registra el mensaje de error y la excepción en un registro de eventos (logger) y luego finaliza la aplicación JavaFX.

LoginController:

- 1. La clase está anotada con @Controller, lo que la marca como un componente controlador en el contexto de Spring.
- 2. Implementa la interfaz Initializable, lo que significa que tiene un método initialize() que se llama cuando se inicializa el controlador.
- 3. La clase tiene anotaciones @FXML en algunos de sus campos, lo que indica que están vinculados a elementos de la interfaz de usuario definidos en un archivo FXML.
- 4. Hay métodos getPassword() y getUsername() que devuelven el contenido de los campos de contraseña (PasswordField) y nombre de usuario (TextField), respectivamente.
- 5. Hay un método login() que se ejecuta cuando se produce un evento de clic en el botón de inicio de sesión (btnLogin). Este método verifica las credenciales de inicio de sesión utilizando el servicio UserService y, si las credenciales son válidas, cambia a una nueva escena utilizando el StageManager.
- 6. La clase tiene una inyección de dependencia del servicio UserService y del StageManager utilizando la anotación @Autowired.
- 7. La clase tiene un método initialize() que se ejecuta al inicializar el controlador. Sin embargo, en este caso, el método está vacío y no contiene ninguna lógica específica.7
- 8. PetController:
- 9. La clase está anotada con @Controller, lo que la marca como un componente controlador en el contexto de Spring.
- 10. Implementa la interfaz Initializable, lo que significa que tiene un método initialize() que se llama cuando se inicializa el controlador.
- 11. La clase tiene anotaciones @FXML en algunos de sus campos, lo que indica que están vinculados a elementos de la interfaz de usuario definidos en un archivo FXML.

- 12. Hay métodos para manejar eventos de clic en los botones "Logout" y "Reset" y para guardar los detalles de una mascota.
- 13. Hay métodos para mostrar alertas de información y validación.
- 14. La clase tiene inyecciones de dependencia del StageManager y del PetService utilizando la anotación @Autowired.
- 15. Hay listas observables (ObservableList) que se utilizan para almacenar la lista de mascotas y los roles.
- 16. Hay métodos para obtener los valores de los campos de la interfaz de usuario, como el nombre, el apellido, la fecha de nacimiento, el género, el rol, el correo electrónico y la contraseña
- 17. Hay un método setColumnProperties() que configura las propiedades de las columnas de una tabla (TableView).
- 18. Hay un método loadPetsDetails() que carga los detalles de las mascotas en la tabla.
- 19. Hay métodos de validación para validar los campos de la interfaz de usuario.
- 20. Hay un método onClickReturn() que se ejecuta cuando se produce un evento de clic en un botón y cambia a una nueva escena utilizando el StageManager.

FXML:

- 1. La enumeración define tres valores: USER, LOGIN y PET, cada uno de los cuales representa una vista específica en la aplicación.
- 2. Cada valor enum tiene dos métodos abstractos: getTitle() y getFxmlFile(). Estos métodos deben ser implementados por cada valor enum para proporcionar el título y la ubicación del archivo FXML asociados con esa vista.
- 3. La implementación de cada valor enum sobrescribe los métodos getTitle() y getFxmlFile() para devolver el título y la ubicación correctos basados en un archivo de recursos (Bundle) que contiene las cadenas correspondientes.
- 4. Hay un método getStringFromResourceBundle() que se utiliza para obtener una cadena específica del archivo de recursos mediante una clave.

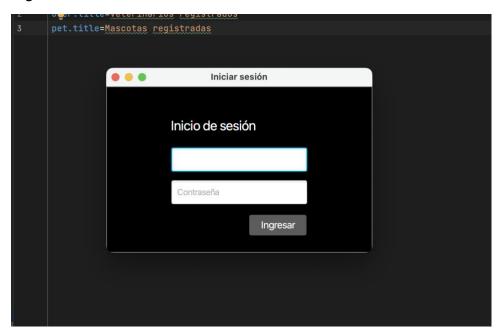
Conexión a la base de datos:

La conexión a la base de datos la hace mediante el application propietes que proporciona springboot, ahí se declara el puerto al que se va a conectar, el usuario y contraseña, se utilizó JPA, el cual hace el mapeo de las clases.

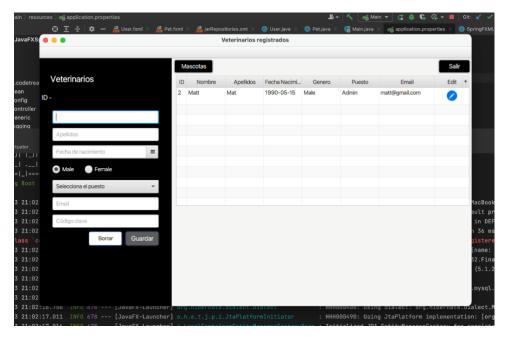
```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/javafx
spring.datasource.username=gerente
spring.datasource.password=1234
```

Capturas del funcionamiento del sistema:

Login:



Veterinarios registrados:



Mascotas registradas:

