

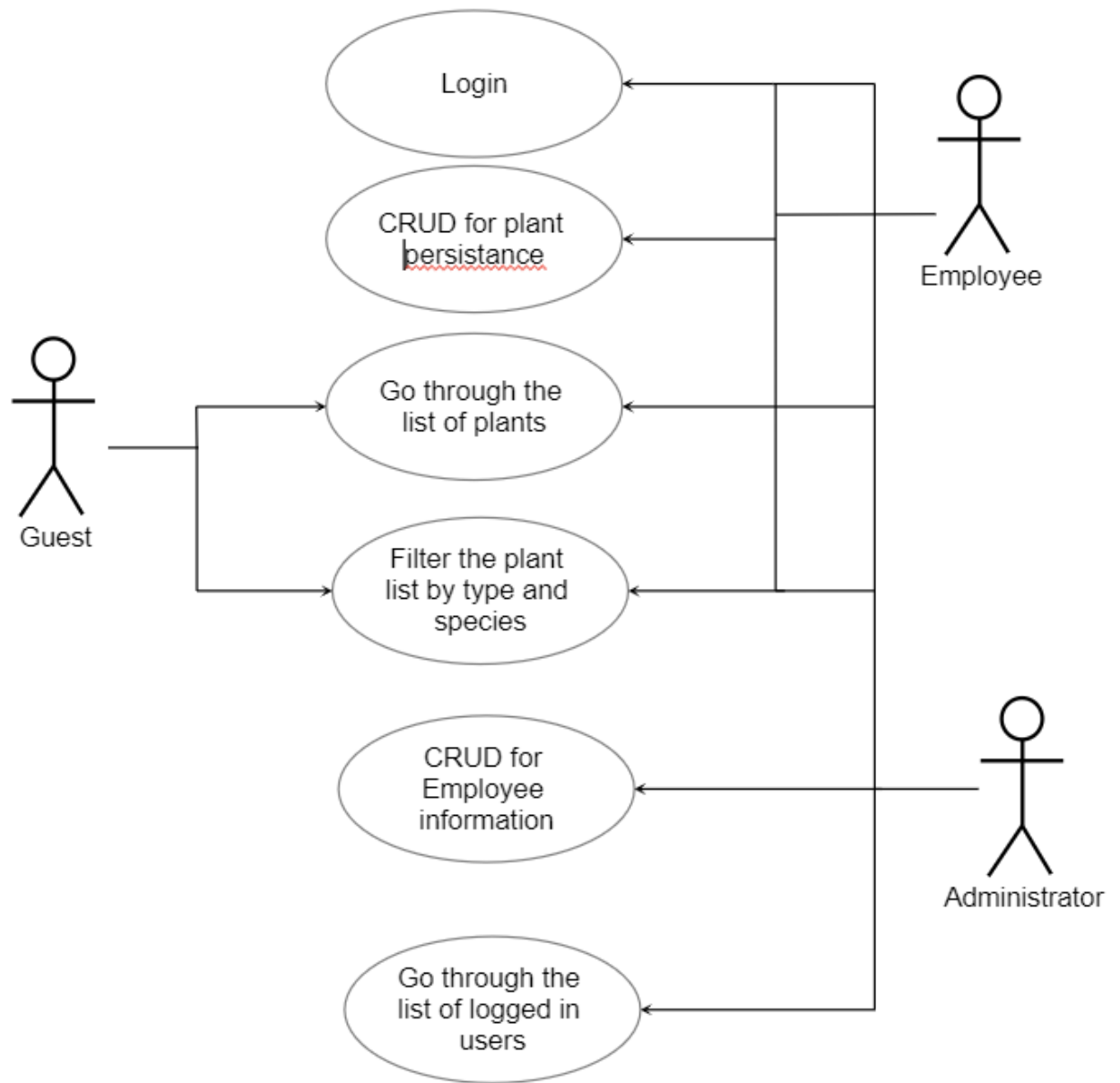
Botanical Garden

Student: Baciú Maria-Simina

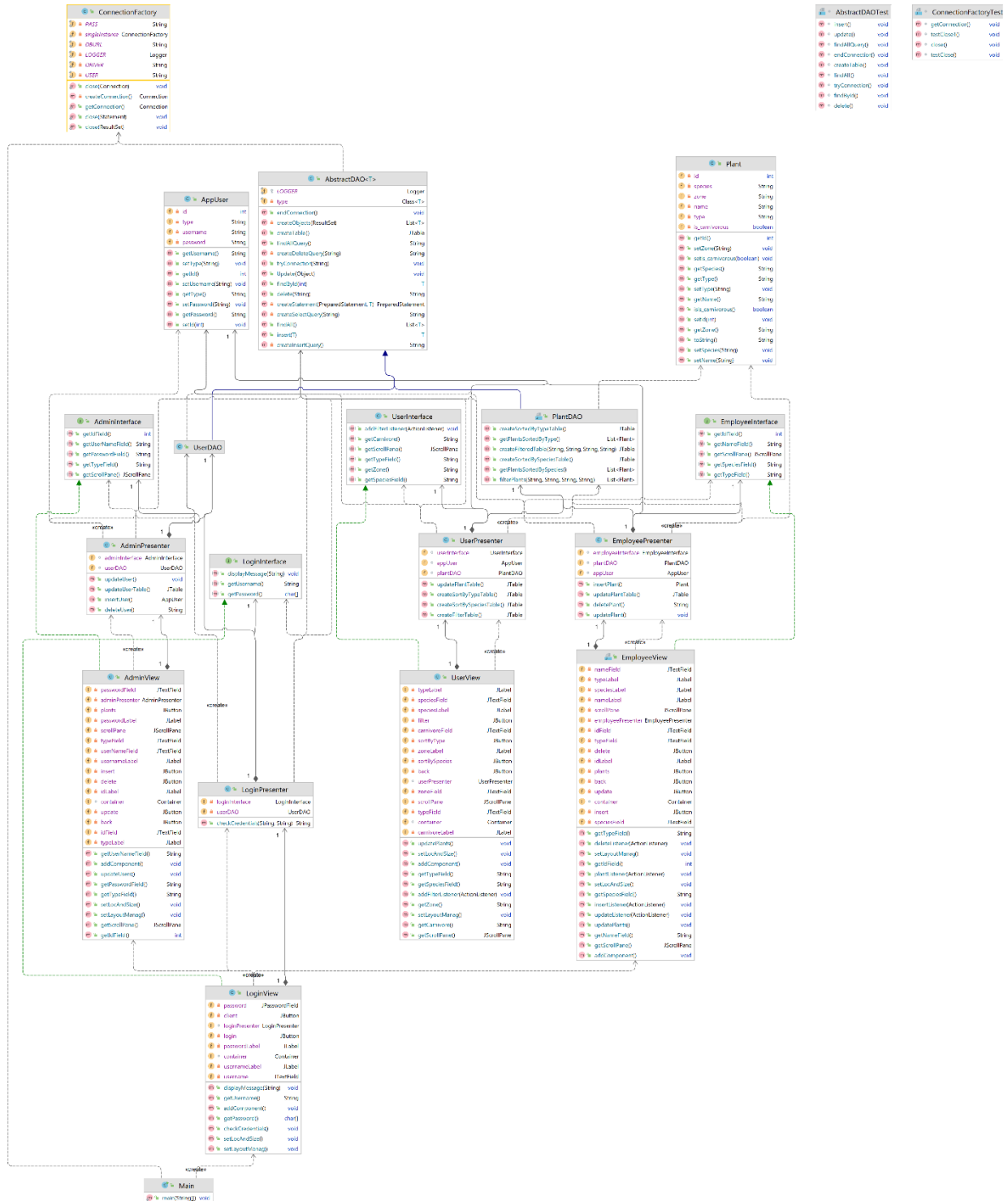
Group: 30434/1

1. Diagrams







1.1 Use case







1.2 Class Diagram



1.3 Entity-Relation

plant	
 id	integer
 name	varchar
 type	varchar
 species	varchar
 zone	varchar
 is_carnivorous	boolean

appuser	
 id	integer
 username	varchar
 password	varchar
 type	varchar

2. Problem

Develop an application that can be used in a botanical garden. The application will have 3 types of users: botanical garden visitor, botanical garden employee, and administrator.

Visitor users can perform the following operations without authentication:

- ❖ Viewing the list of all plants in the botanical garden sorted by type and species;

- ❖ Filtering the plant list by the following criteria: type, species, carnivorous plants, botanical garden zone.

Botanical garden employee users can perform the following operations after authentication:

- ❖ All operations allowed for visitor users;
- ❖ CRUD operations regarding the persistence of plants in the botanical garden.

Administrator users can perform the following operations after authentication:

- ❖ All operations allowed for visitor users;
- ❖ CRUD operations for information related to users who require authentication;
- ❖ Viewing the list of users who require authentication.

3. Used Technologies

For the database I have chosen to use PostgreSQL. PostgreSQL, often called Postgres, is a powerful, open-source relational database management system (RDBMS) that is known for its scalability, reliability, and extensive features. Postgres supports SQL and offers advanced features such as complex queries, indexing, transactions, and more. It is also highly extensible and offers support for custom data types, functions, and procedures. Postgres is a popular choice for data-driven applications and provides excellent support for concurrent access and data consistency.

As for the GUI (graphical user interface) I have chose to work with Java Swing as it was the tool for graphic interfaces that I've used before. Java Swing is a set of GUI components for Java applications. It is a part of the Java Foundation Classes (JFC) and provides a rich set of components for creating desktop applications that run on a variety of platforms. Java Swing provides a wide range of components such as buttons, labels, text fields, etc.

4. Language

For a programming language I have chosen Java, as it is the programming language that I know best. A few more reasons I have chosen to use Java are because I have already implemented a project that had a connection to a database so I knew that it would not be as complicated, as well as I knew how to implement the GUIs (which I believe that in another language would have taken me a lot more time to implement).

5. Description of UML

The LoginView class is associated with the LoginPresenter class through a dependency relationship. This means that the LoginView class uses the LoginPresenter class, but the LoginPresenter class does not depend on the LoginView class.

The UserView class is associated with the UserPresenter class through a dependency relationship. This means that the UserView class uses the UserPresenter class, but the UserPresenter class does not depend on the UserView class.

The EmployeeView class is associated with the EmployeePresenter class through a dependency relationship. This means that the EmployeeView class uses the EmployeePresenter class, but the EmployeePresenter class does not depend on the EmployeeView class.

The AdminView class is associated with the AdminPresenter class through a dependency relationship. This means that the AdminView class uses the AdminPresenter class, but the AdminPresenter class does not depend on the AdminView class.

The UserView class implements the UserInterface interface. This means that it has to implement the methods defined in the UserInterface interface.

The EmployeeView class implements the EmployeeInterface interface. This means that the EmployeeView class has to implement the methods defined in the EmployeeInterface interface.

6. Description of app

1. ConnectionFactory

This is a Java class for creating database connections using JDBC. It uses the Singleton design pattern to ensure that only one instance of the class is created.

The class has private constants for the database driver, URL, username, and password. These constants are used to create the connection to the database.

The *createConnection()* method is used to create a new connection to the database. It uses the constants to create the connection.

The *getConnection()* method is a public static method that returns the connection object. It calls the *createConnection()* method to create a new connection.

The *close()* methods are used to close the connection, statement, and result set objects. These methods are used to release the resources used by the database connection.

Overall, this class provides a convenient and reusable way to create database connections in a Java application.

2. AbstractDAO

This is a generic DAO (Data Access Object) class that provides common database operations such as inserting, updating, and deleting objects, as well as retrieving objects from the database. The class uses generics to allow for type-safe access to the database tables.

The class has a constructor that determines the type of object that will be handled by the DAO. The *createSelectQuery()* method is used to create a query for selecting all fields based on a specific ID. The *findAllQuery()* method creates a query to retrieve all objects from the database table.

The *findAll()* method retrieves all objects from the database and returns a list of objects.

The *findById()* method retrieves an object from the database based on the specified ID.

The *createTable()* method creates a JTable with the data from the database table. The *insert()* method inserts an object into the database table. The *Update()* method updates an object in the database table. The *delete()* method deletes an object from the database table.

Overall, this class provides a convenient and reusable way to access database tables in a Java application.

3. PlantDAO

This is a Java class that represents a data access object (DAO) for the Plant class. It provides methods for retrieving and manipulating plant data from a database or other data source.

The class extends an abstract DAO class and overrides its generic methods to work specifically with the Plant class. It also includes additional methods for sorting and filtering plant data, as well as creating JTable objects for displaying plant data in a graphical user interface.

getPlantsSortedByType(): Retrieves all plants from the data source and sorts them by type in ascending order.

getPlantsSortedBySpecies(): Retrieves all plants from the data source and sorts them by species in ascending order.

filterPlants(String type, String species, String zone, String carnivore): Retrieves all plants from the data source and filters them based on the specified criteria (type, species, zone, and carnivorousness).

createFilteredTable(String type, String species, String zone, String carnivore): Calls the *filterPlants* method to retrieve filtered plant data, creates a JTable object based on the data, and returns the table.

createSortedBySpeciesTable(): Calls the *getPlantsSortedBySpecies* method to retrieve sorted plant data, creates a JTable object based on the data, and returns the table.

createSortedListByTypeTable(): Calls the *getPlantsSortedListByType* method to retrieve sorted plant data, creates a *JTable* object based on the data, and returns the table.

4. UserDao

This is a Java class that defines a DAO (Data Access Object) for plants.

The class extends the *AbstractDAO* class, which provides a basic implementation of the CRUD (Create, Read, Update, Delete) operations that are common to most data access objects.

5. AppUser

This class represents a user in an application.

It has four private instance variables: "id", "username", "password", and "type".

The class contains getter and setter methods for each of the instance variables. These methods allow other parts of the application to access and modify the values of the instance variables.

6. Plant

This class represents a plant object and contains attributes related to it such as its name, type, species, zone, and whether it is carnivorous or not. It has a constructor to initialize these attributes, and getter and setter methods to retrieve and modify them.

7. AdminPresenter

This is a Java class called *AdminPresenter*. It is used to handle user interactions with an *AdminInterface*, which is not shown in the code. The *AdminPresenter* uses a *UserDAO* object to interact with the data layer and perform CRUD (create, read, update, delete) operations on the user data.

The *AdminPresenter* has the following methods:

updateUserTable(): returns a *JTable* containing all the user data retrieved from the *UserDAO* object.

insertUser(): creates a new *AppUser* object with the data entered by the user in the *AdminInterface*, and adds it to the database using the *UserDAO* object. Returns the newly created *AppUser* object.

deleteUser(): deletes an *AppUser* object from the database using the *UserDAO* object. The user data is obtained from the *AdminInterface*. Returns a *String* indicating the success or failure of the delete operation.

updateUser(): updates an existing AppUser object in the database using the UserDao object. The updated data is obtained from the AdminInterface.

Overall, the AdminPresenter acts as an intermediary between the user interface and the data layer, allowing the user to interact with the user data in a user-friendly way while maintaining the integrity of the data.

8. EmployeePresenter

This is the implementation of the EmployeePresenter class. This presenter is responsible for handling the logic and communication between the EmployeeInterface view and the PlantDAO data access object.

It has methods to update the plant table displayed in the UI, to insert a new plant record, to delete a plant record, and to update an existing plant record. These methods receive input data from the EmployeeInterface, create a Plant object with the received data, and then pass it to the PlantDAO for further processing.

The *updatePlantTable()* method fetches all plant records from the database through the PlantDAO, creates a table model with the data, and returns a JTable object that can be used to display the data in the UI.

The *insertPlant()* method creates a new Plant object with data from the input fields of the EmployeeInterface, and then passes it to the PlantDAO for insertion into the database.

The *deletePlant()* method creates a new Plant object with the ID of the record to be deleted, and passes it to the PlantDAO for deletion.

The *updatePlant()* method creates a new Plant object with the data from the input fields of the EmployeeInterface, and then passes it to the PlantDAO for update.

Overall, the EmployeePresenter serves as the intermediary between the UI and the data access layer, allowing for separation of concerns and easier maintenance of the code.

9. LoginPresenter

This is a presenter class named LoginPresenter which is responsible for handling the login functionality. It contains a constructor that takes an instance of LoginInterface as a parameter and initializes an instance of UserDao. It has a single public method named *checkCredentials* which takes the username and password as parameters and returns a

String representing the user type if the user is found in the database and the password matches, otherwise it returns null.

Overall, the LoginPresenter class facilitates the authentication process by checking the user credentials entered by the user in the login form against the credentials stored in the database.

10. UserPresenter

This is the presenter for the user interface, which allows users to view and filter plants. It has several methods that interact with the PlantDAO to retrieve data and create tables. The *createFilterTable()* method calls the PlantDAO's *createFilteredTable()* method, passing in filter criteria entered by the user via the user interface. The *createSortByTypeTable()* and *createSortBySpeciesTable()* methods call the PlantDAO's *getPlantsSortedByType()* and *getPlantsSortedBySpecies()* methods, respectively, to retrieve lists of plants sorted by the specified criteria. Finally, *updatePlantTable()* calls the PlantDAO's *createTable()* method to create a table of all plants in the database.

11. AdminInterface

This is an interface that defines the methods required for the AdminInterface view. It has the following methods:

getIdField(): returns an integer value corresponding to the input in the ID field.

getUserNameField(): returns a string value corresponding to the input in the username field.

getPasswordField(): returns a string value corresponding to the input in the password field.

getTypeField(): returns a string value corresponding to the input in the type field.

getScrollPane(): returns a JScrollPane object that can be used to display a table in the view.

12. AdminView

The AdminView class is responsible for implementing the AdminInterface and building the graphical user interface (GUI) for the admin user. The GUI contains several input fields, labels, and buttons, which are all defined as private member variables. The container object is used to hold all the GUI components.

The AdminPresenter object is instantiated and initialized in the constructor. The updateUser method is used to update the JTable with data from the database. The addComponent method is used to add listeners to the buttons and set the layout and positioning of the GUI components. The getIdField, getUsernameField, getPasswordField, getTypeField, and getScrollPane methods are used to retrieve input data and scroll pane for the table.

Overall, the AdminView is responsible for displaying and updating user data for the admin user.

13. EmployeeInterface

The EmployeeInterface interface defines the methods that should be implemented by any class that wants to act as the view for an employee in the plant nursery management system. The methods are:

getIdField(): returns an integer representing the ID entered by the employee in the corresponding text field.

getNameField(): returns a string representing the name entered by the employee in the corresponding text field.

getTypeField(): returns a string representing the type entered by the employee in the corresponding text field.

getSpeciesField(): returns a string representing the species entered by the employee in the corresponding text field.

getScrollPane(): returns a JScrollPane component that displays a table of plants in the nursery.

14. EmployeeView

The EmployeeView class is a GUI interface that extends JFrame and implements EmployeeInterface. It has several attributes that are UI components like JTextField, JButton, JScrollPane, and JLabel. The class has a constructor that sets the layout manager, sets the location and size of the UI components, and adds components to the container. The class also has a method *updatePlants()* that updates the table of plants displayed in the view. This method calls the corresponding method in EmployeePresenter class that retrieves the table from the database and updates the table in the view.

The class has four action listeners that respond to user actions like inserting, deleting, updating plants, and going back to the previous view. These action listeners are attached to the corresponding buttons in the UI. When the user clicks on the button, the action

listener invokes the corresponding method in `EmployeePresenter` that updates the plants in the database.

The class also has several getter methods that return the values of the UI components like the ID, name, type, and species of the plant, and a `JScrollPane` object that displays the table of plants.

15. LoginInterface

This is an interface for the login view. It includes methods for displaying a message, getting the username, and getting the password.

16. LoginView

The code is implementing the view for the login screen of the plant management system. It defines an interface called `LoginInterface` that has three methods: `displayMessage()` which displays a message to the user, `getUsername()` which gets the entered username from the text field, and `getPassword()` which gets the entered password from the password field.

The `LoginView` class extends `JFrame` and implements the `LoginInterface`. It defines the components for the login screen, such as the username and password fields, login button, and client button. It sets their size and location on the screen. It also adds action listeners to the login and client buttons.

The `checkCredentials()` method checks if the entered username and password match a valid user in the system. If the credentials are correct, it determines the type of user and opens the appropriate view for that user. If the credentials are incorrect, it displays an error message to the user.

17. UserInterface

The `UserInterface` interface specifies the methods that a class representing a user interface for the plant viewing system should implement. It includes methods for adding a filter listener, getting the filter criteria (zone and carnivore), getting the plant type and species, and getting the `JScrollPane` for displaying the plant table.

18. UserView

The `UserView` interface provides methods for the `UserView` class, which represents the view for the clients. It defines methods for adding a filter listener, getting the values of

various fields such as zone, carnivore, type, and species, and getting the scroll pane to display the results. The class also has a method called `updatePlants` which updates the plant table with the latest data.