Implementation of a Software Component Management Subsystem

[48103] Cláudia Seabra
[47020] Francisco Martins
[48117] Joana Nogueira
[47857] Kecty Jenifer

]

Informatics Engineering
Web Technologies Laboratory
Prof Dr. Thiago Andrade, Prof Dr. Bruno Cunha,
Prof. Dr.ª Maria João Ferreira

09/06/2024

UPT DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

CONTENT

INTRODUCTION

This report was developed as part of the Software Engineering and Web Technologies Laboratory project. It focused on architectural design and improving the learning of HTML, CSS, JavaScript, and PHP.

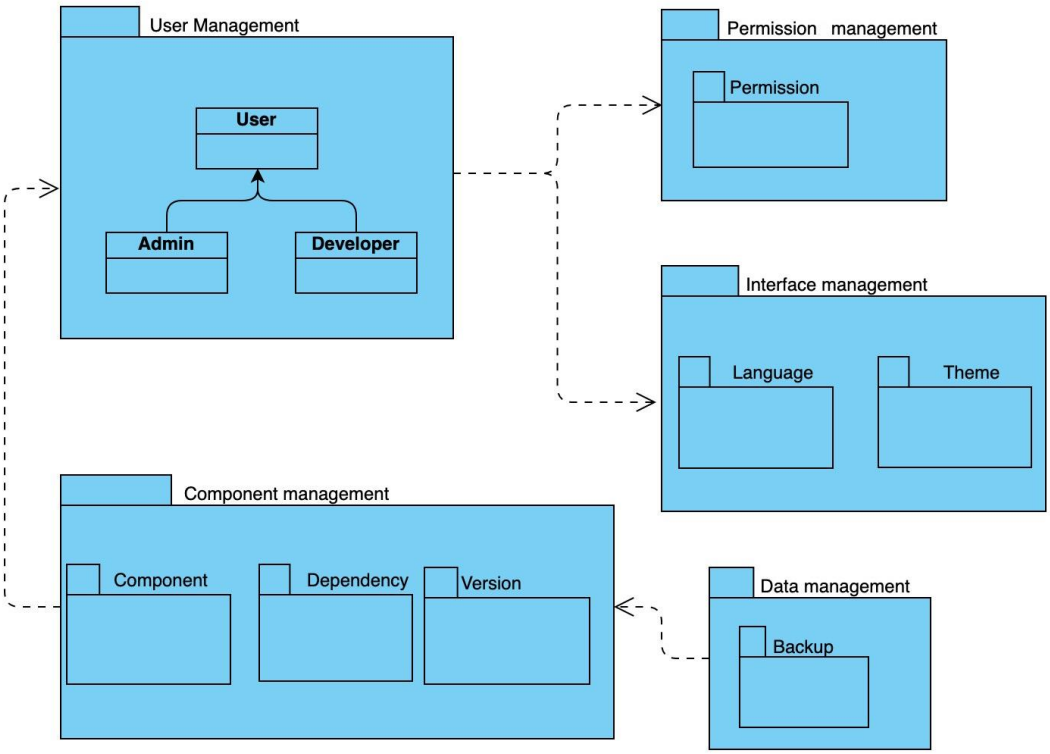
The objective was to create a software component management system, starting with the interface development for administrators and developers to implement the presented use cases. The interface was enhanced using CSS to make it more appealing and intuitive. Additionally, PHP was used to connect to the database, enabling the updating and usage of data from it.

Finally, it is important to mention that package diagrams, class diagrams, hybrid diagrams, and the ER model were developed. These were fundamental in understanding the system and achieving a more efficient implementation.

SOFTWARE ENGINEERING

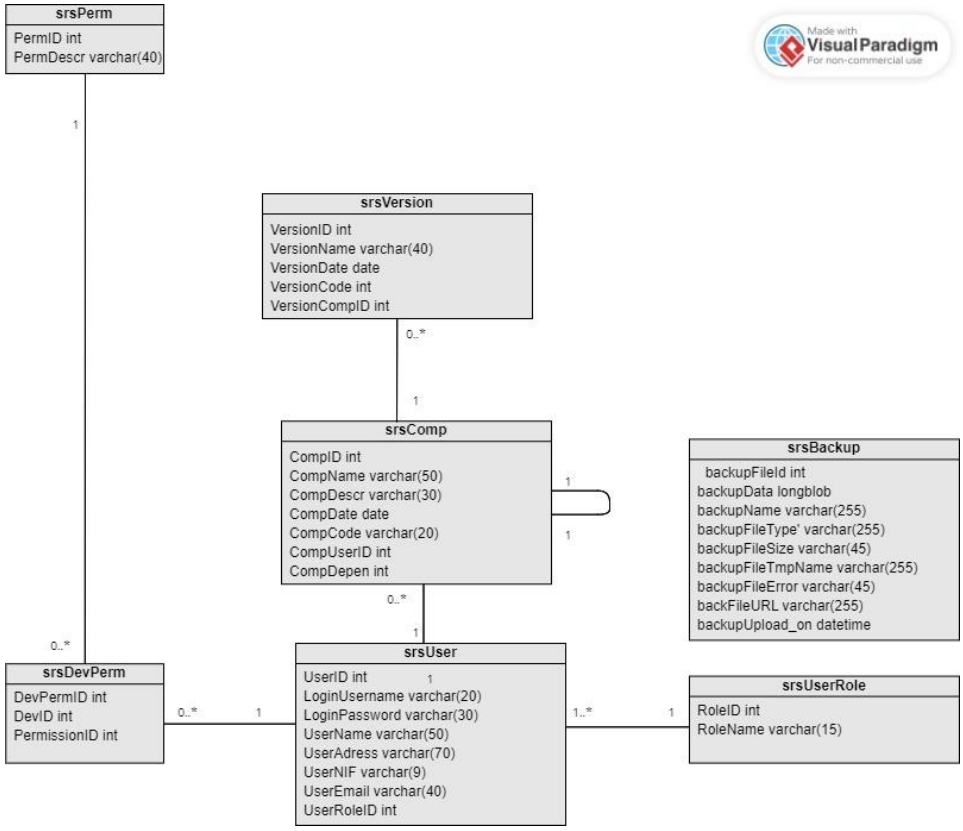
1. PACKAGE DIAGRAM

The diagram below illustrates the packages and their relationships within a software component management system. Package diagrams are helpful in simplifying complex class diagrams and structuring classes into packages, with the goal of representing and organizing the different elements and their dependencies.



2. CLASS DIAGRAM

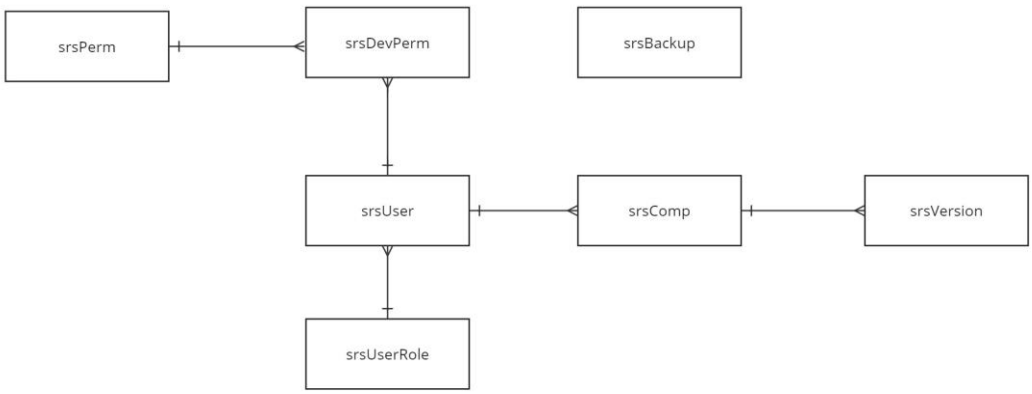
Class diagrams serve as tools to delineate the structure of a system by portraying its classes, attributes, operations, and inter-object relationships. It provides valuable insight as each class in the diagram corresponds to a database table, offering a comprehensive representation of the system.



3. ENTITY-RELATIONSHIP MODEL

After creating the class diagram, it was converted into the ER model. This conversion provided a visual representation of how the entities are linked, how the relationships function, and where improvements can be made.

The image below illustrates the ER model that has been created for the system developed.



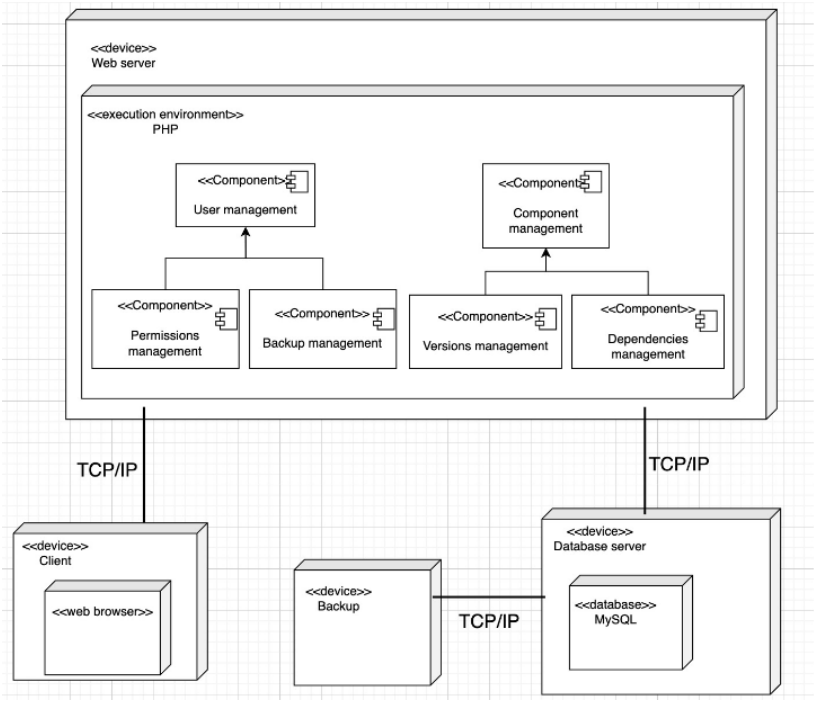
4. HYBRID DIAGRAM

The hybrid diagram is a visual representation that combines elements such as use cases, sequences, and classes. Its purpose is to provide a comprehensive view of the system.

In the system in question, it details how the different software components within the PHP environment interact with each other, as well as with other devices such as the client, database server, and backup device.

In an in-depth analysis, the PHP execution environment runs within the web server and contains the main components of the software management system. Additionally, within the environment, there is the User Management component, which connects to Permissions Management, responsible for managing user permissions, and Backup Management, which handles user data backup and restoration operations. Following the same logic, the management component generates the components stored in the system and connects to Versions Management, which handles different versions of software components, and Dependencies Management, which manages dependencies between them.

In conclusion, the client device connects to the web server via the TCP/IP protocol, allowing developers and administrators to interact with the system. The database server stores system data, including information about users, components, versions, and dependencies, and connects to the web server via TCP/IP to provide and receive data. Additionally, it has a connection to the backup device, where data is stored for recovery in case a backup is needed.



5. INTERFACE DESIGN

In this stage of interface design, we initiated the process to gain a clearer understanding of our project's appearance.

We began by applying fundamental principles to design a user interface and selected an accessible and intuitive interface. All of our interfaces are graphic user interfaces (GUIs), and we have incorporated color themes and language options to enhance the user experience. Additionally, we began outlining user cases to understand the workflow for each operation. In conclusion, when everything is brought together, the website becomes more accessible and easier to manage.

ADMIN

1. Register User

Name:	Register user
Actor:	Admin
Short description:	The use case allows the admin to register a new user.
Precondition:	Admin must be logged in
Normal flow sequence:	1. The Use Case starts when the admin selects the option “Register user”. 2. The system displays the entry form details for the new user. 3. Admin enters the new user information. 4. The system validates the new user.
Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu.
Postcondition:	There is a new user in the system.

This user diagram begins at interface number 2 and then goes to interface number 3. If the user is successfully registered, it will return to interface number 2.

2. Edit User

Name:	Edit user
Actor:	Admin
Short description:	The use case allows the admin to edit a user.
Precondition:	Admin must be logged in and there must be at least a user registered.
Normal flow sequence:	1. The Use Case starts when the admin selects the option “Edit user”. 2. The system displays the entry form.

	3. The system displays the entry form for the details the admin wants to change from the user. 4. The system validates the editing of user data.
Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu. If the user data is invalid the system shows that the user was not found.
Postcondition:	The user has been edited.

The user diagram begins at interface number 2, and then proceeds to interface number 4. If the user is not found, it will move to interface number 5; otherwise, it will proceed to interface number 13. If the user is successfully edited, it will return to interface number 2.

3. Deactivate User

Name:	Deactivate user
Actor:	Admin
Short description:	The use case allows the admin to deactivate a user.
Precondition:	Admin must be logged in and there must be at least a user registered.
Normal flow sequence:	1. The Use Case starts when the admin selects the option “Deactivate user”. 2. The system displays the entry form. 3. Admin enters the name of the user that wants to be removed. 4. The system validates the removal of the user.
Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu. If the user data is invalid the system shows that the user was not found.
Postcondition:	The user has been removed.

This user diagram begins at interface number 2, passes to interface number 6, and cycles back to interface number 2 if the user is removed.

4. Register Permission

Name:	Register permission
Actor:	Admin
Short description:	The use case allows the admin to register a new permission.
Precondition:	Admin must be logged in.

Normal flow sequence:	1. The Use Case starts when the admin selects the option “Register permission”. 2. The system displays the entry form. 3. Admin enters the description of the new permission. 4. The system validates the new permission.
Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu.
Postcondition:	There is a new permission in the system.

This user diagram begins at interface number 2 and then moves to interface number 7. If the permission is already registered, interface number 8 will appear; otherwise, it will return to interface number 2.

5. Add Permission

Name:	Add permission
Actor:	Admin
Short description:	The use case allows the admin to add a permission for a user.
Precondition:	Admin must be logged in and there must be at least a developer registered, and a permission registered.
Normal flow sequence:	1. The Use Case starts when the admin selects the option “Change permissions”. 2. The system displays the entry form. 3. Admin enters the username of the user. 4. The system displays the list of all permissions that the system has and the permissions that the user has. 5. The system displays an entry form. 6. Admin enters the permission ID to be added to a user. 7. The system validates the addition of permission to a user.
Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu. If the user data is invalid the system shows that the user was not found.
Postcondition:	Permission added.

The user diagram begins at interface number 2, then moves to interface number 9. If the user exists, it will proceed to interface number 10. Upon successful permission addition, it will return to interface number 2.

6. Remove Permission

Name:	Remove permission
Actor:	Admin
Short description:	The use case allows the admin to remove a permission from a user.
Precondition:	Admin must be logged in and there must be at least a developer registered and permission associated with a developer.
Normal flow sequence:	1. The Use Case starts when the admin selects the option “Change permissions”. 2. The system displays the entry form. 3. Admin enters the username of the user. 4. The system displays the list of all permissions that the system has and the permissions that the user has. 5. The system displays an entry form. 6. Admin enters the permission ID to be removed from a user. 7. The system validates the removal of the user.
Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu. If the user data is invalid the system shows that the user was not found.
Postcondition:	Permission removed.

This user diagram starts at interface number 2 and then progresses to interface number 9. If the user exists, it will proceed to interface number 10. If the permission is successfully removed, it will return to interface number 2.

7. Backup

Name:	Backup
Actor:	Admin
Short description:	The use case allows the admin to backup the system.
Precondition:	Admin must be logged in.
Normal flow sequence:	1. The Use Case starts when the admin selects the option “Backup”. 2. The system displays a button to download the backup. 3. Admin clicks the button to download the backup. 4. The system creates a new backup on the database.

Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu.
Postcondition:	There is a new backup in the system.

This user diagram begins at interface number 2 and then proceeds to interface number 1. If the backup is successful, it will return to interface number 2.

8. Restore Backup

Name:	Restore backup
Actor:	Admin
Short description:	The use case allows the admin to restore a backup to the system.
Precondition:	Admin must be logged in and there must be at least a backup to restore.
Normal flow sequence:	1. The Use Case starts when the admin selects the option “Restore backup”. 2. The system displays backups created in the database. 3. Admin selects the backup he wants to restore. 4. The system restored the backup that the admin selected.
Alternative flows:	At any time, the admin can cancel the operation and return to the admin menu.
Postcondition:	Backup restored.

This user diagram starts at interface number 2 and then goes to interface number 12. If the backup is restored, it will return to interface number 2.

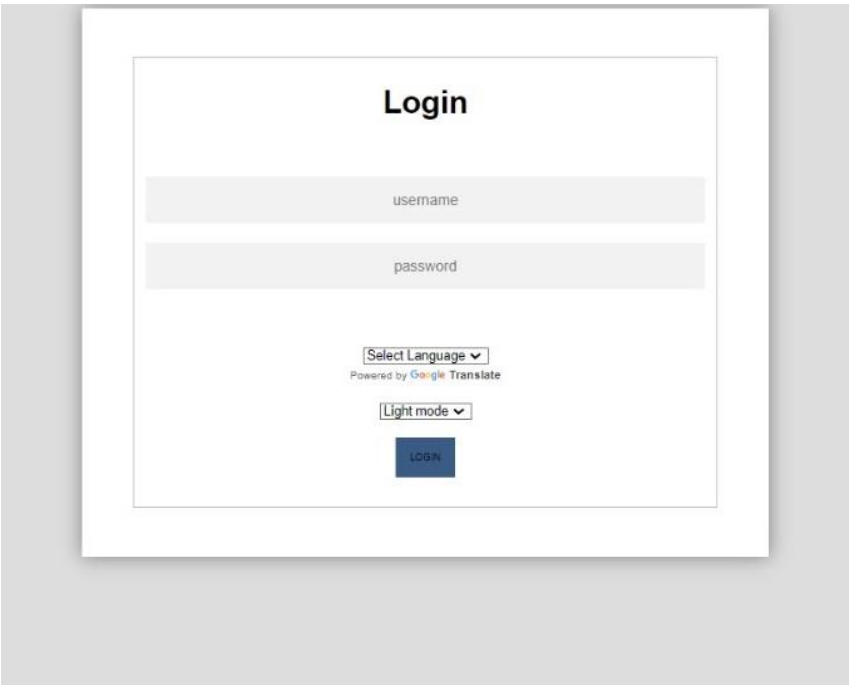
9. Logout

Name:	Logout
Actor:	Admin
Short description:	The use case allows the admin to logout.
Precondition:	Admin must be logged in.
Normal flow sequence:	1- The system displays a button to logout. 2- The admin selects the logout button. 3-The system logs out the admin.
Alternative flows:	
Postcondition:	Admin logged out.

This user diagram begins at interface number 2 and proceeds to number 1 after the user has successfully logged out.

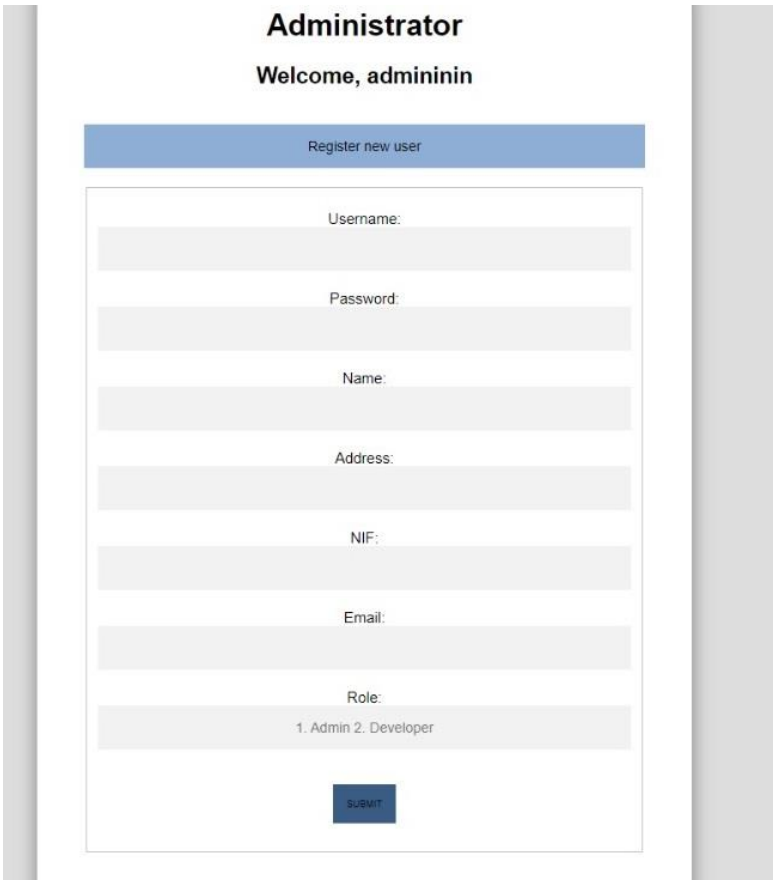
ADMIN'S INTERFACES

1. Login



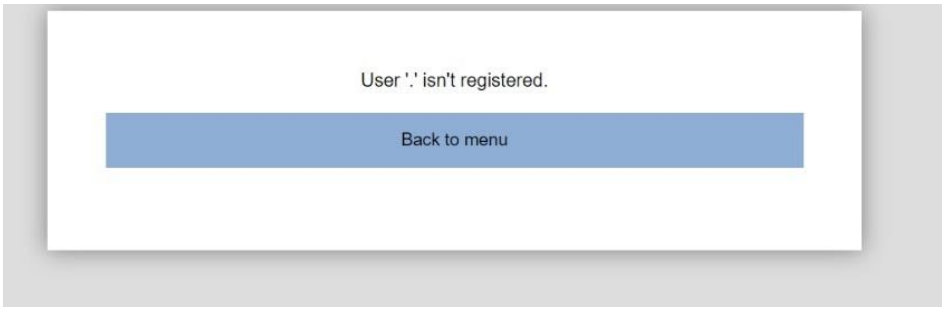
2. Admin User

3. Register New User



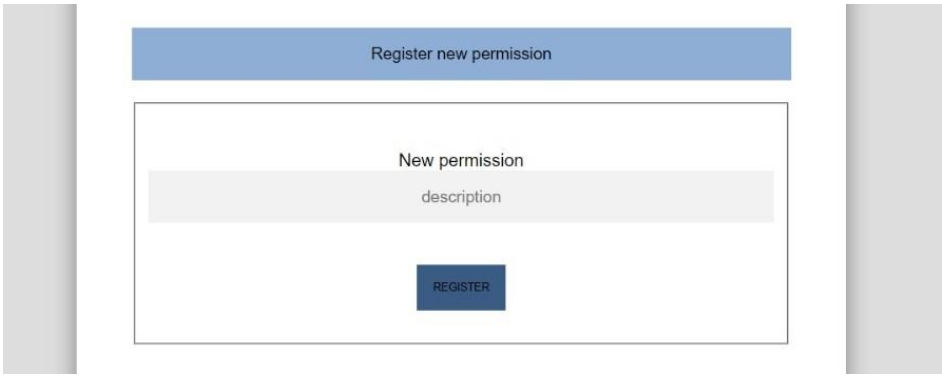
4. Edit User

5. User Not Found

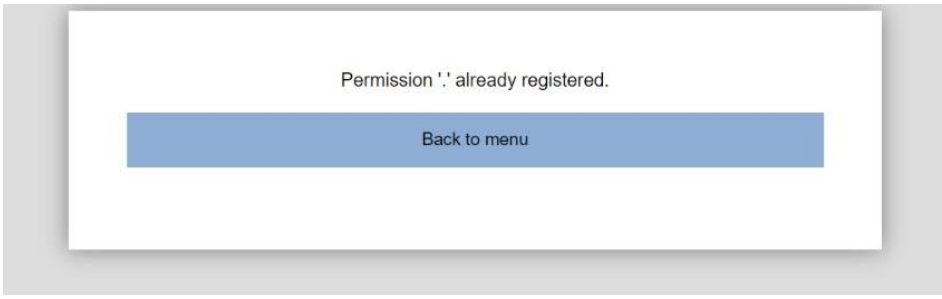


6. Deactivate User

7. Register New Permission

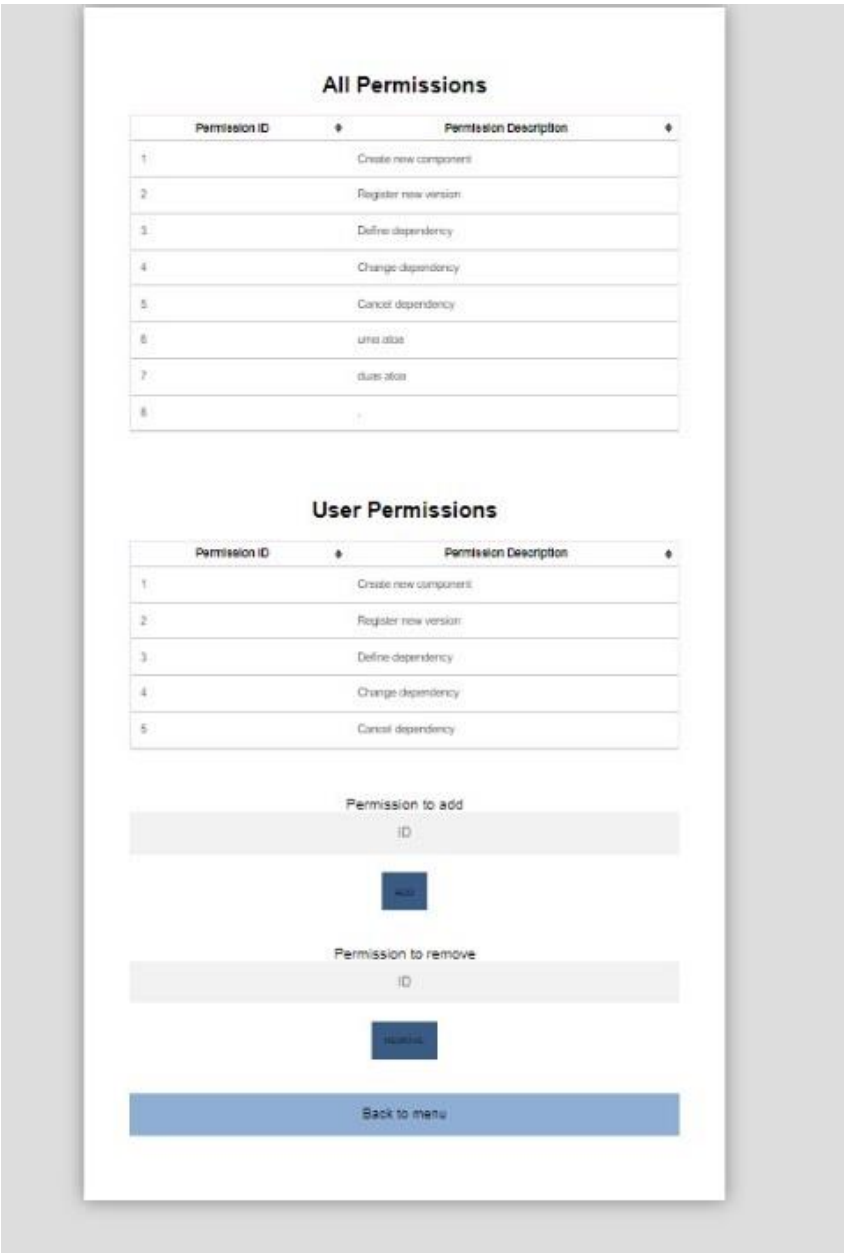


8. Register New Permission – Permission Already Registered

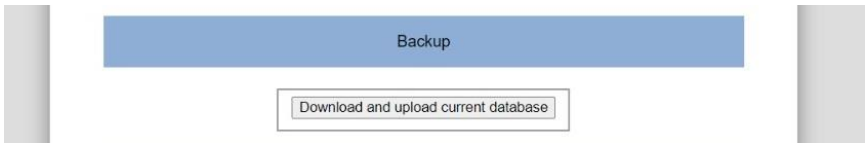


9. Change developer's permission

10. Change developer's permissions form

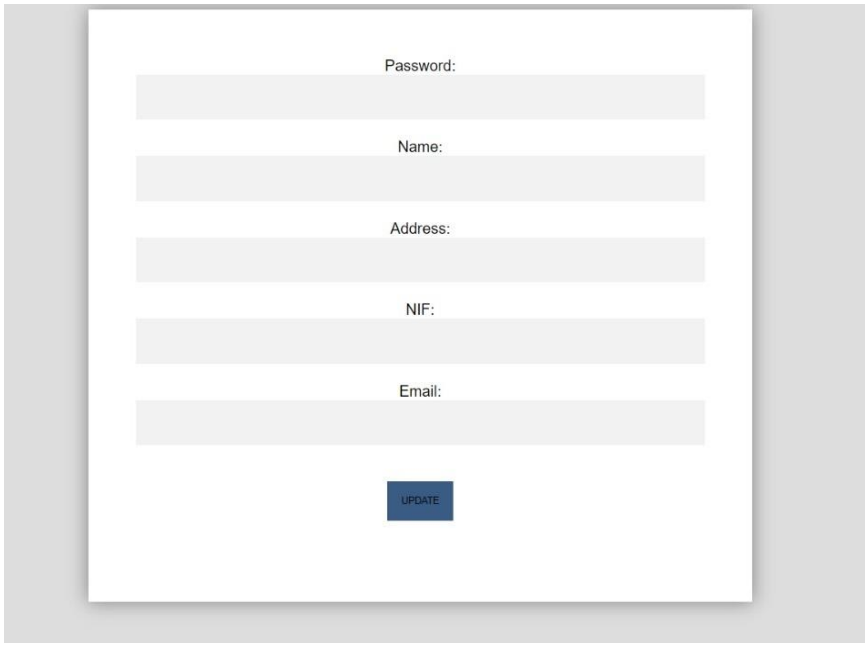


11. Backup



12. Restore backup

13. Edit User Form



DEVELOPER

1. Register new component

Name:	Register new component
Actor:	Developer
Short description:	The use case allows the developer to register a new component.
Precondition:	The developer must be logged in.
Normal flow sequence:	1. The Use Case starts when the developer selects the option “Register new component”. 2. The system displays the entry form details for the new component. 3. The developer enters the new component information. 4. The system validates the new component.
Alternative flows:	At any time the developer can cancel the operation and return to the developer menu.
Postcondition:	There is a new component in the system.

The user diagram begins at interface number 1 and then proceeds to interface number 12. If the user is successfully registered, it will return to interface number 1; otherwise, it will proceed to interface number 13. At any point, the user can return to interface number 1.

2. Define dependency

Name:	Define dependency
Actor:	Developer
Short description:	The use case allows the developer to define the dependency of a component.
Precondition:	The developer must be logged in and there must be components.
Normal flow sequence:	1. The Use Case starts when the developer selects the option “Define dependency”. 2. The system displays the entry form details for the dependency and a list of components. 3. The developer enters the component ID for the dependency and the dependent component ID. 4. The system validates the dependen-

	cies.
Alternative flows:	At any time, the developer can cancel the operation and return to the developer menu. If the component or the dependency data is invalid, the system displays that the dependency cannot be defined. If the component already has a dependency, a message will be displayed indicating that.
Postcondition:	There is a new dependency in the system.

This diagram starts at interface number 1, then moves to interface number 2, followed by interface number 3. If the component is successfully registered, it returns to interface number 1.

3. Change dependency

Name:	Change dependency
Actor:	Developer
Short description:	The use case allows the developer to change the dependency of a component.
Precondition:	The developer must be logged in and there must be at least two components registered.
Normal flow sequence:	1. The Use Case starts when the developer selects the option “Change dependency”. 2. The system displays the entry form details for the dependency. 3. The developer enters the component ID for the dependency and the dependent component ID. 4. The system validates the changing of the dependency.
Alternative flows:	At any time, the developer can cancel the operation and return to the developer menu. If the component or the dependency data is invalid, the system displays that the dependency cannot be defined. If the component already has a dependency, a message will be displayed indicating that.
Postcondition:	Dependency has been changed.

The user diagram begins at interface number 1, then moves on to interface number 2, followed by interface number 5. If the component is successfully changed, it will return to interface number 1.

4. Remove dependency

Name:	Remove dependency
Actor:	Developer
Short description:	The use case allows the developer to remove the dependency of a component.
Precondition:	The developer must be logged in and there must be a dependency registered to a component.
Normal flow sequence:	1. The Use Case starts when the developer selects the option “Remove dependency”. 2. The system displays the entry form details for the dependency. 3. The developer enters the component ID to be removed. 4. The system validates the removal of the dependency.
Alternative flows:	At any time, the developer can cancel the operation and return to the developer menu. If the component or the dependency data is invalid, the system displays that the dependency cannot be defined. If the component already has a dependency, a message will be displayed indicating that.
Postcondition:	Dependency has been removed.

This user diagram flows from interface number 1 to interface number 2, and then to interface number 4. If the component is successfully canceled, it will return to interface number 1.

5. Register version

Name:	Register version
Actor:	Developer
Short description:	The use case allows the developer to register a new version of the component.
Precondition:	The developer must be logged in and there must be a component registered.
Normal flow sequence:	1. The Use Case starts when the developer selects the option “Register version”.

	2. The system displays the entry form details for the version and a list of components. 3. The developer enters the version form information needed. 4. The system validates the registered version.
Alternative flows:	At any time, the developer can cancel the operation and return to the developer menu. If the component data is invalid the system displays that the component was not found.
Postcondition:	There is a new version in the system.

This user diagram begins at interface number 1 and then progresses to interface number 15. If the version is successfully registered, it will return to interface number 1; otherwise, it will move to interface number 16. At any point, it can go back to interface number 1.

6. List Versions

Name:	List versions
Actor:	Developer
Short description:	The use case allows the developer to list all component versions.
Precondition:	The developer must be logged in and there must be a version registered.
Normal flow sequence:	1. The Use Case starts when the developer selects the option “List versions”. 2. The system displays the entry form details for the dependency and a list of components. 3. The developer enters the component ID. 4. The system displays a list of versions of the component selected.
Alternative flows:	At any time the developer can cancel the operation and return to the developer menu.
Postcondition:	Listed versions.

The user diagram begins at interface number 1 and then moves on to interface number 6. If the user selects a component with versions, interface number 7 will appear; otherwise, interface number 8 will appear. At any point, the user can also return to interface number 1.

7. Search Component

Name:	Search component
Actor:	Developer
Short description:	The use case allows the developer to search for a component.
Precondition:	The developer must be logged in and there must be a component registered.
Normal flow sequence:	1. The Use Case starts when the developer selects the option “Search component”. 2. The system displays the entry form details for the component. 3. The developer enters the component ID. 4. The system displays the component selected.
Alternative flows:	At any time, the developer can cancel the operation and return to the developer menu. If the component data is invalid the system displays that the component was not found.
Postcondition:	Component found.

The diagram starts at interface number 1 and then moves on to interface number 9. From there, it proceeds to interface number 10 if the component is found, otherwise, it will go to interface number 11. At any point, the developer can go back to interface number 1.

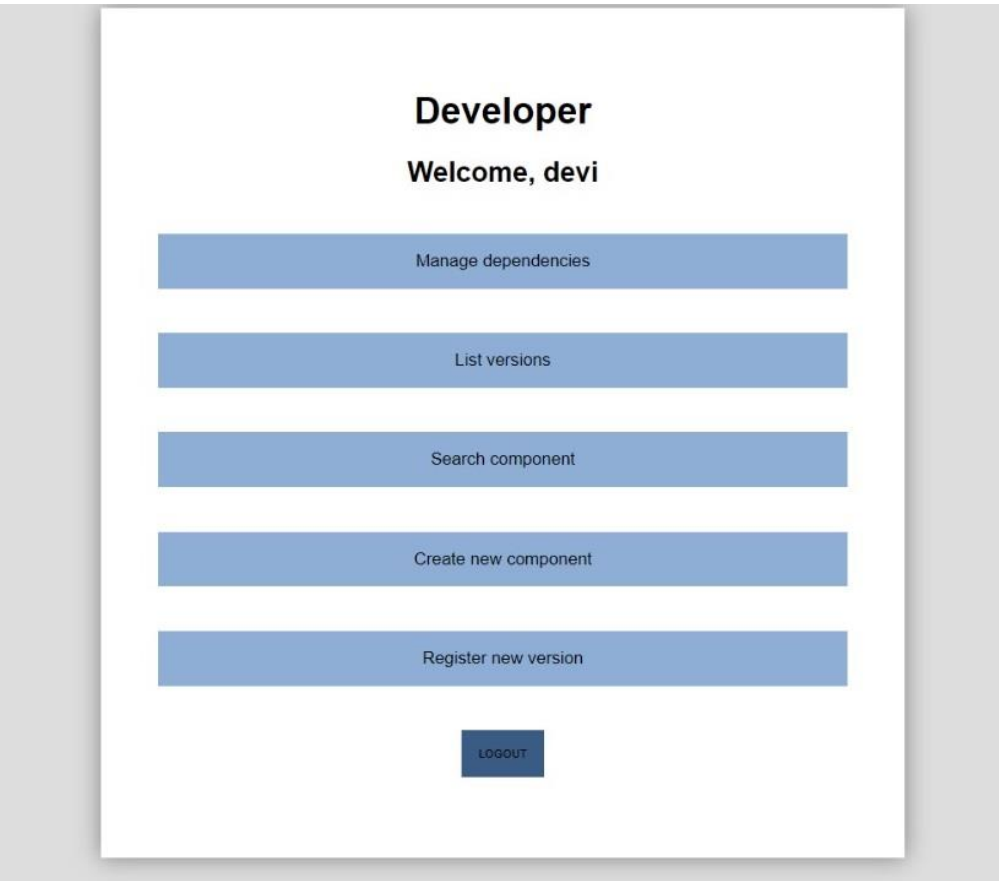
8. Logout

Name:	Logout
Actor:	Developer
Short description:	The use case allows the developer to logout.
Precondition:	The developer must be logged in.
Normal flow sequence:	1. The system displays a button to logout. 2. The developer selects the logout button. 3. The system logs out the developer.
Alternative flows:	
Postcondition:	The developer logged out.

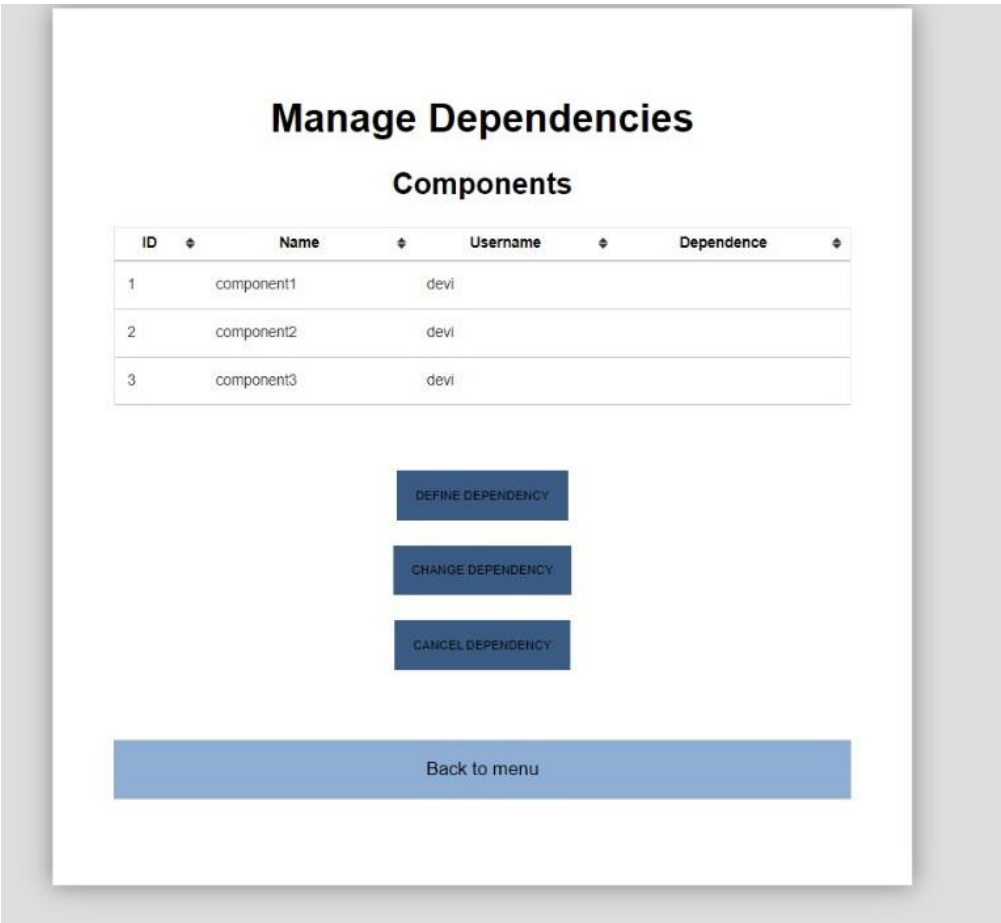
This user diagram starts at interface number 1. If the user successfully logs out, it will transition to interface number 14.

DEVELOPER'S INTERFACES

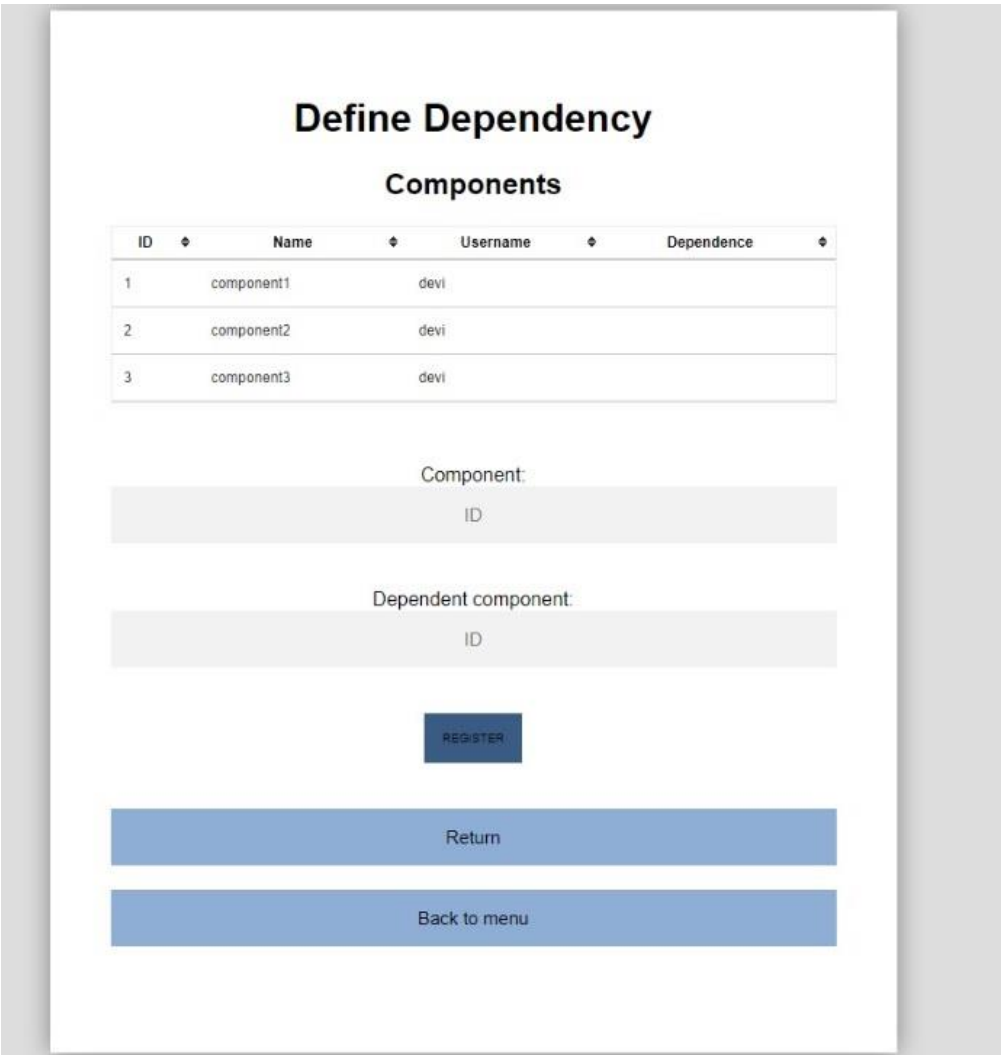
1. Developer Menu



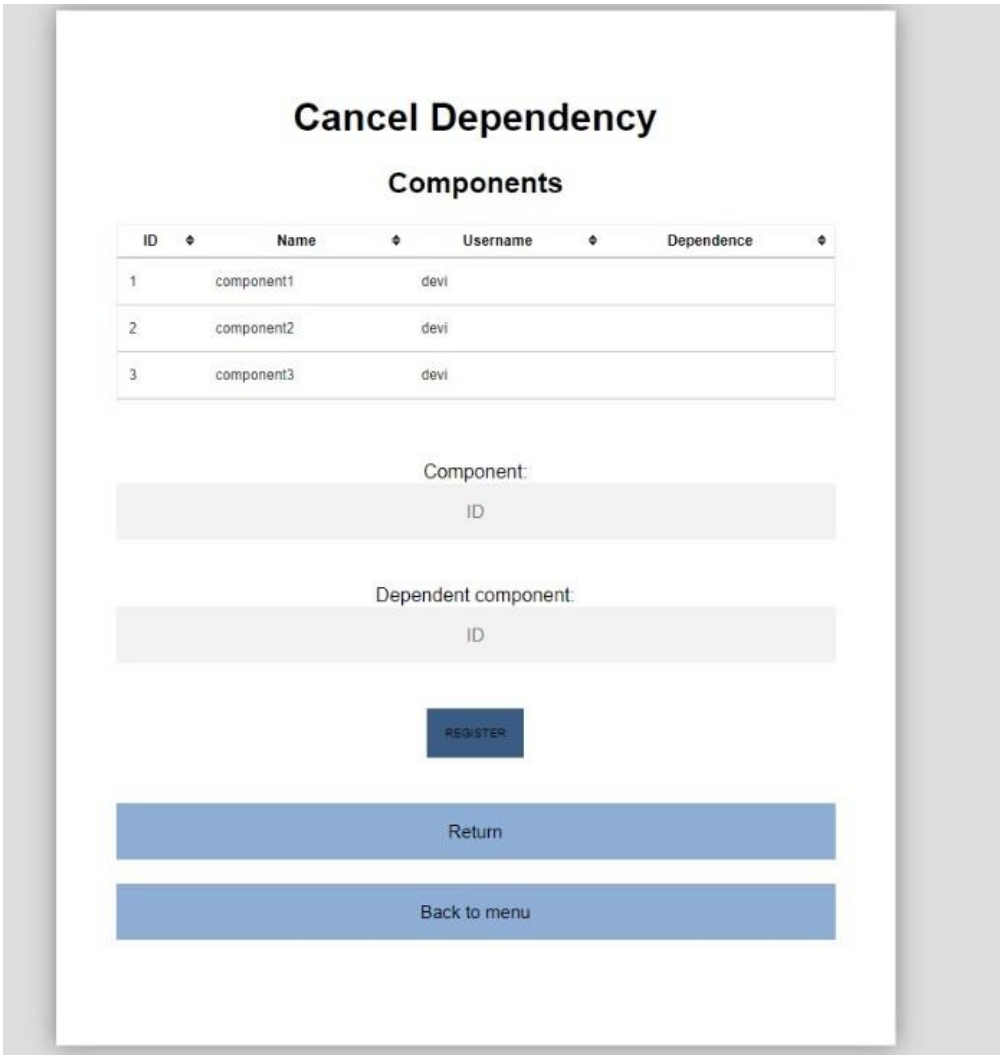
2. Manage dependencies



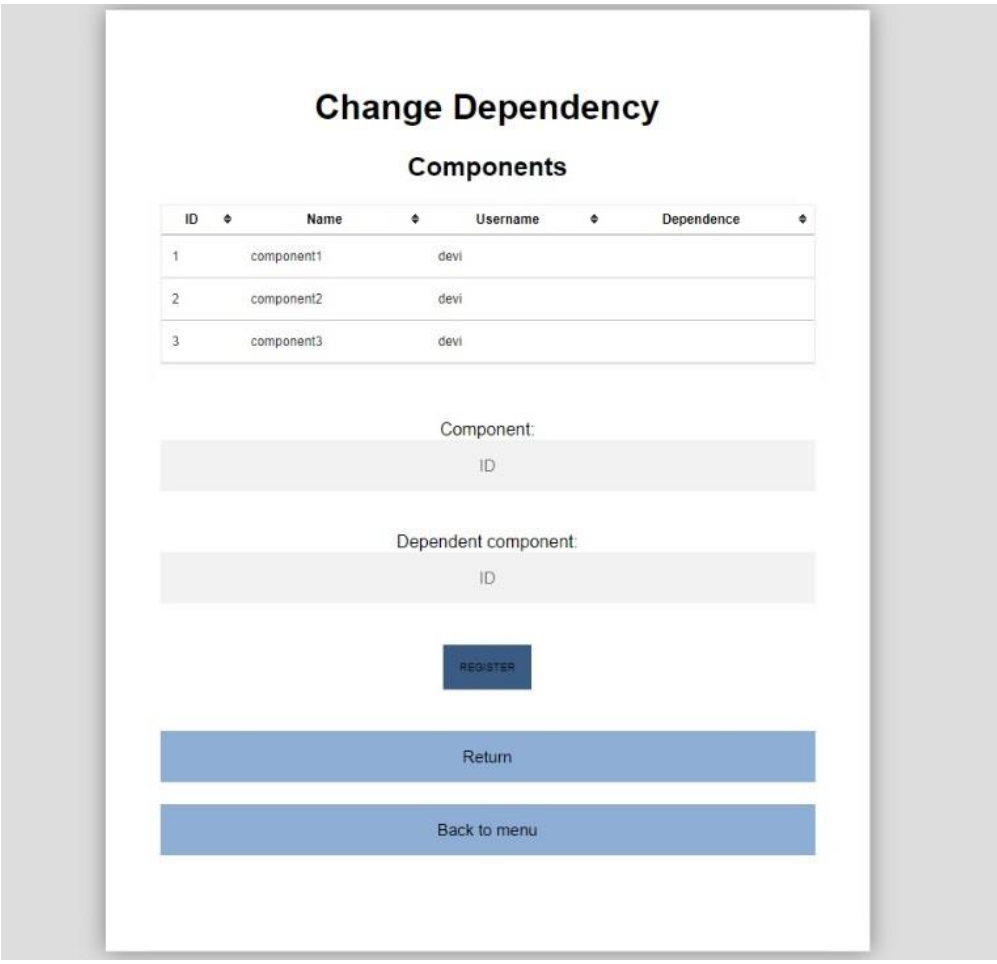
3. Define dependency



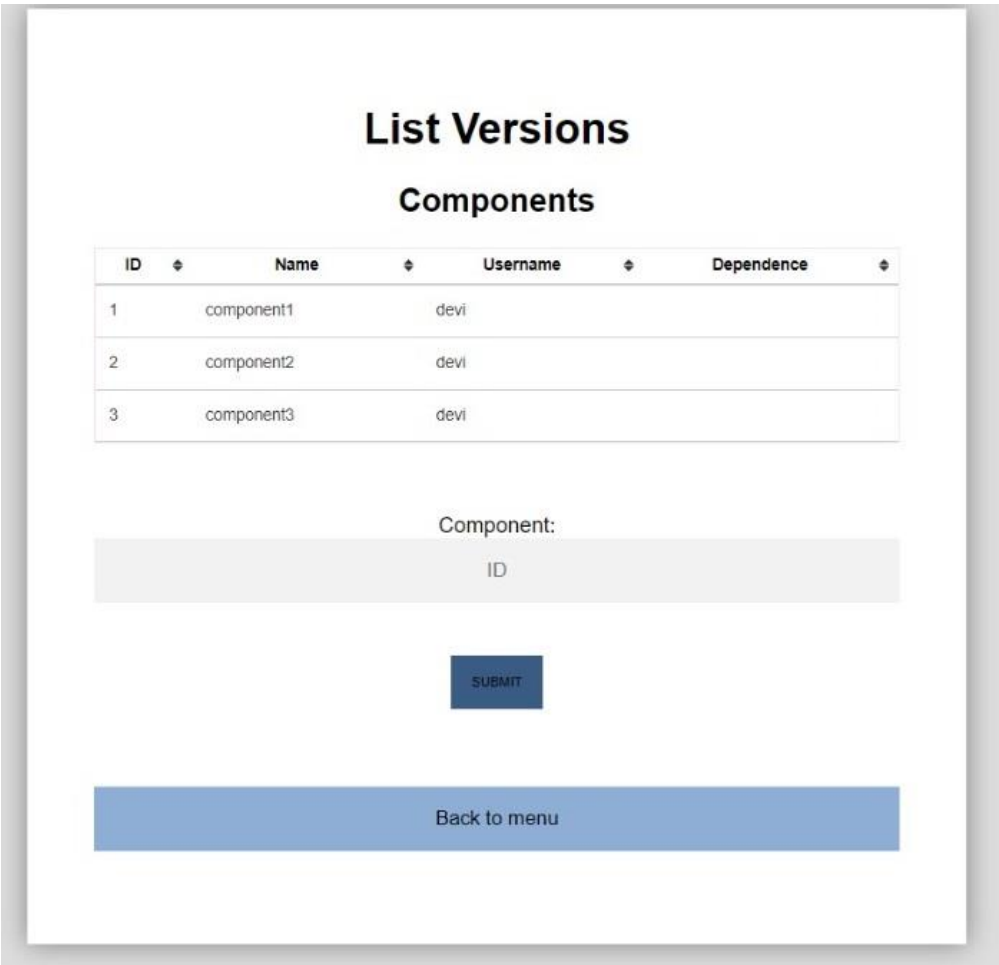
4. Cancel dependency



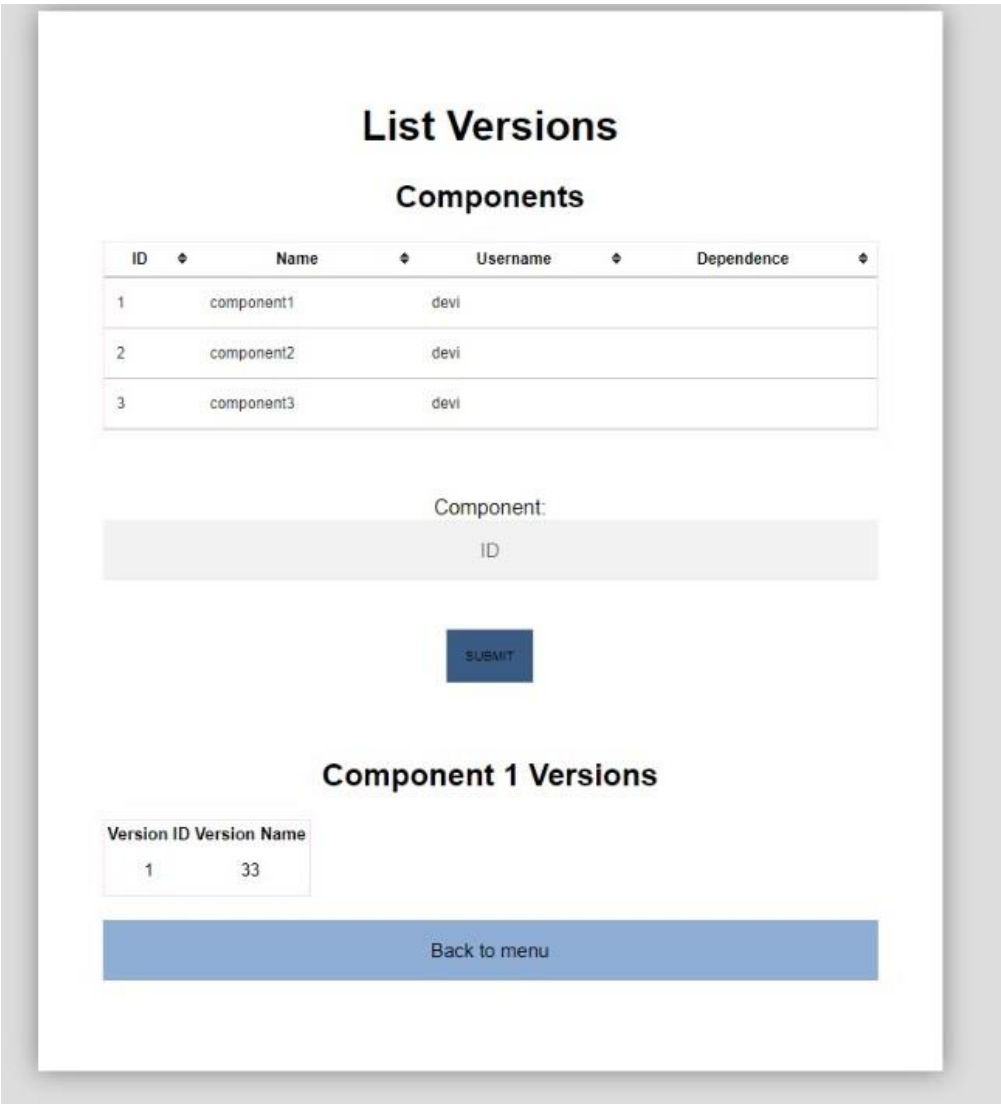
5. Change dependency



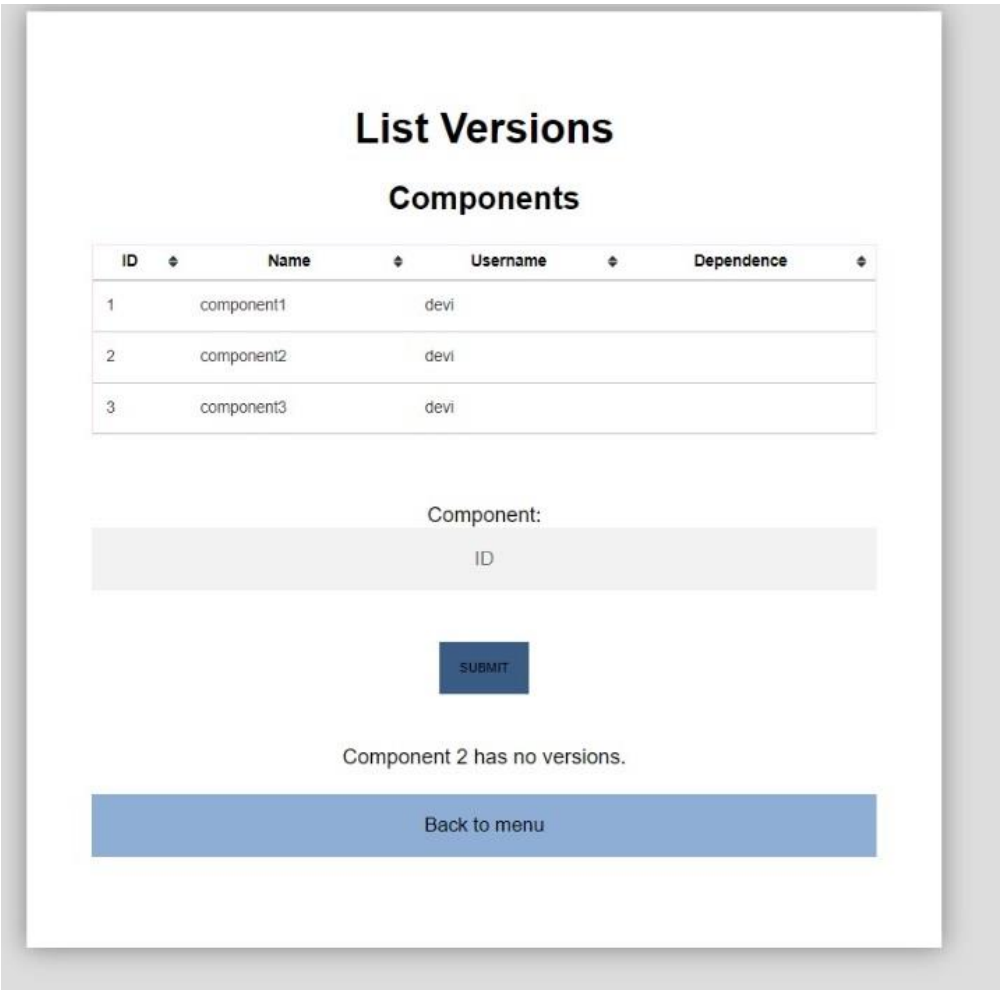
6. List Versions



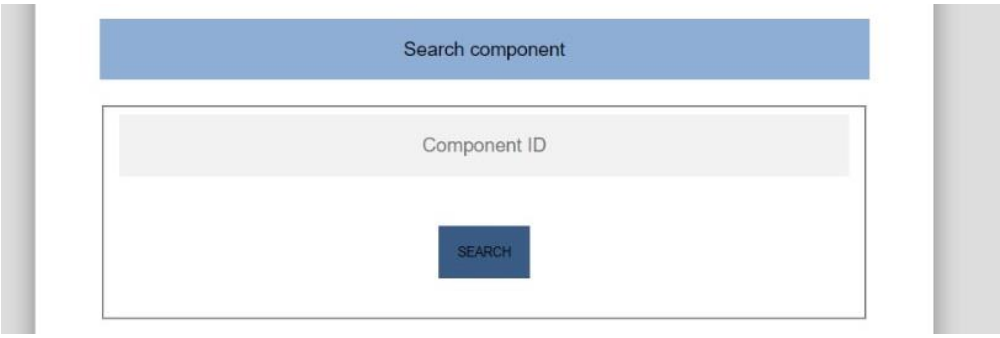
7. List Versions – versions displayed



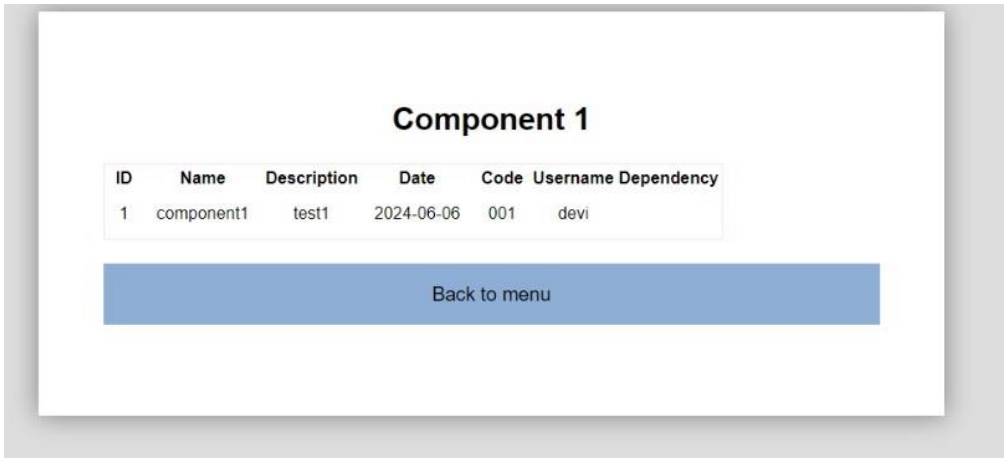
8. List Versions – component without versions



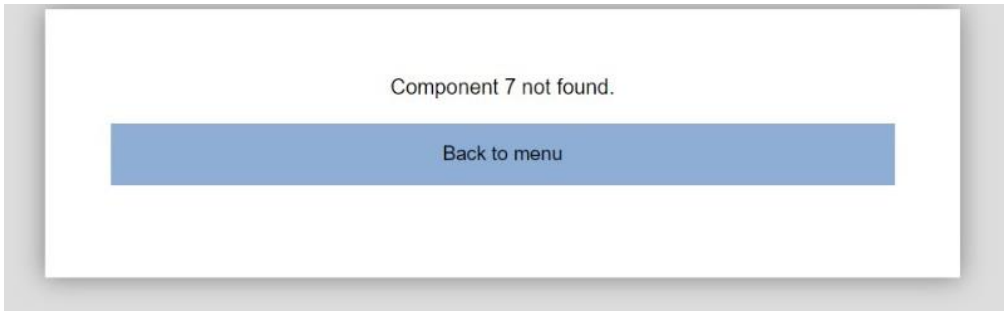
9. Search Component



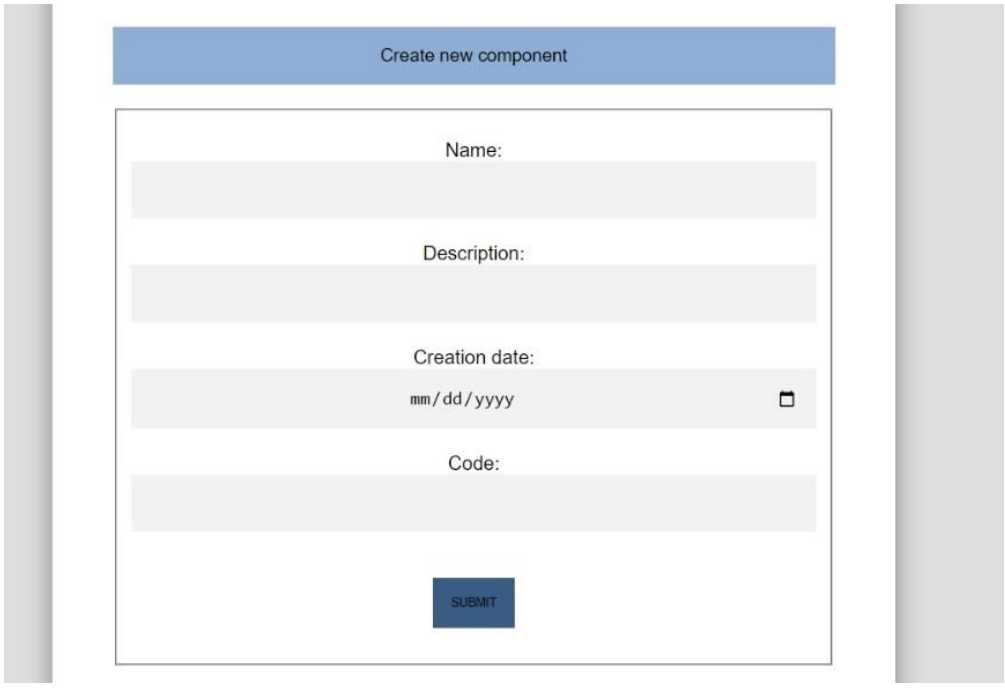
10. Search Component – Component Found



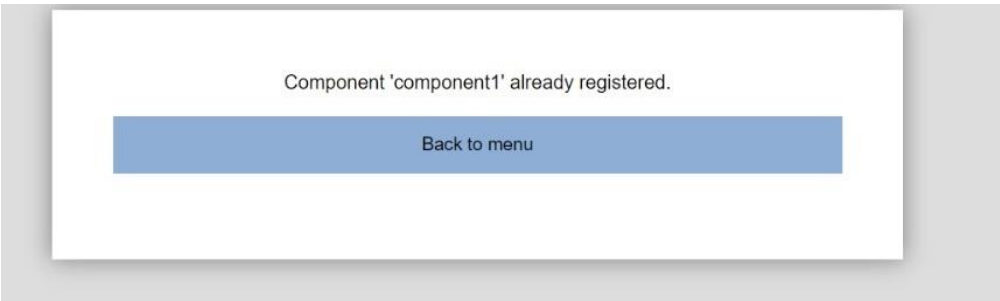
11. Search Component – Component Not Found



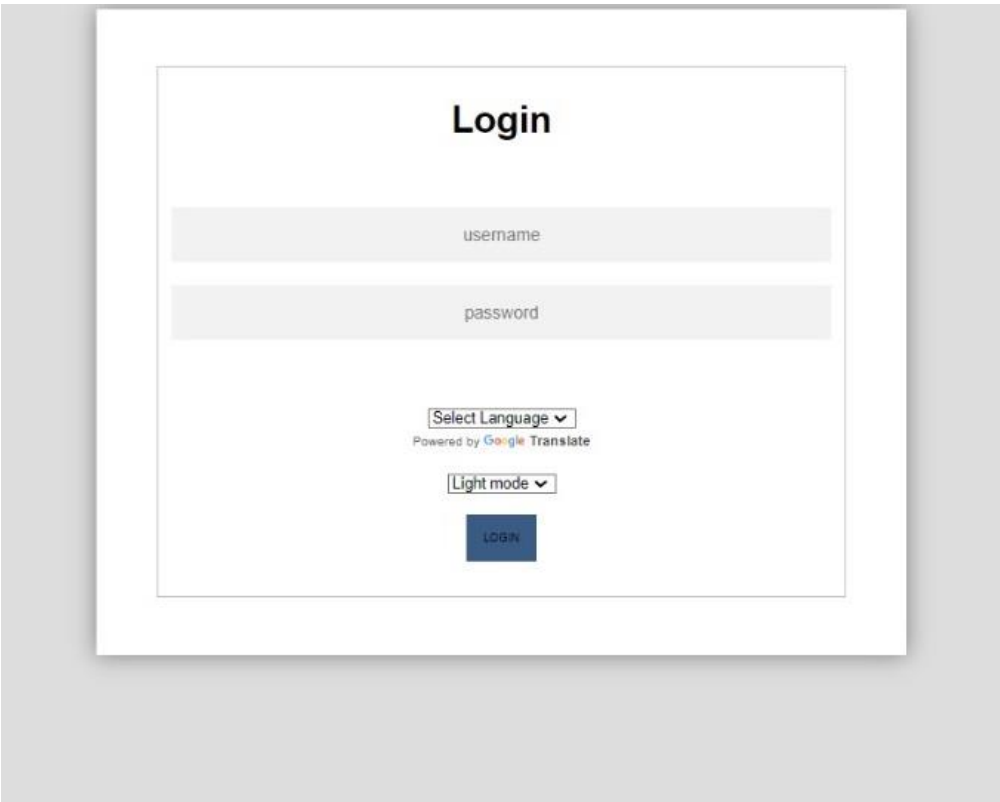
12. Register Component



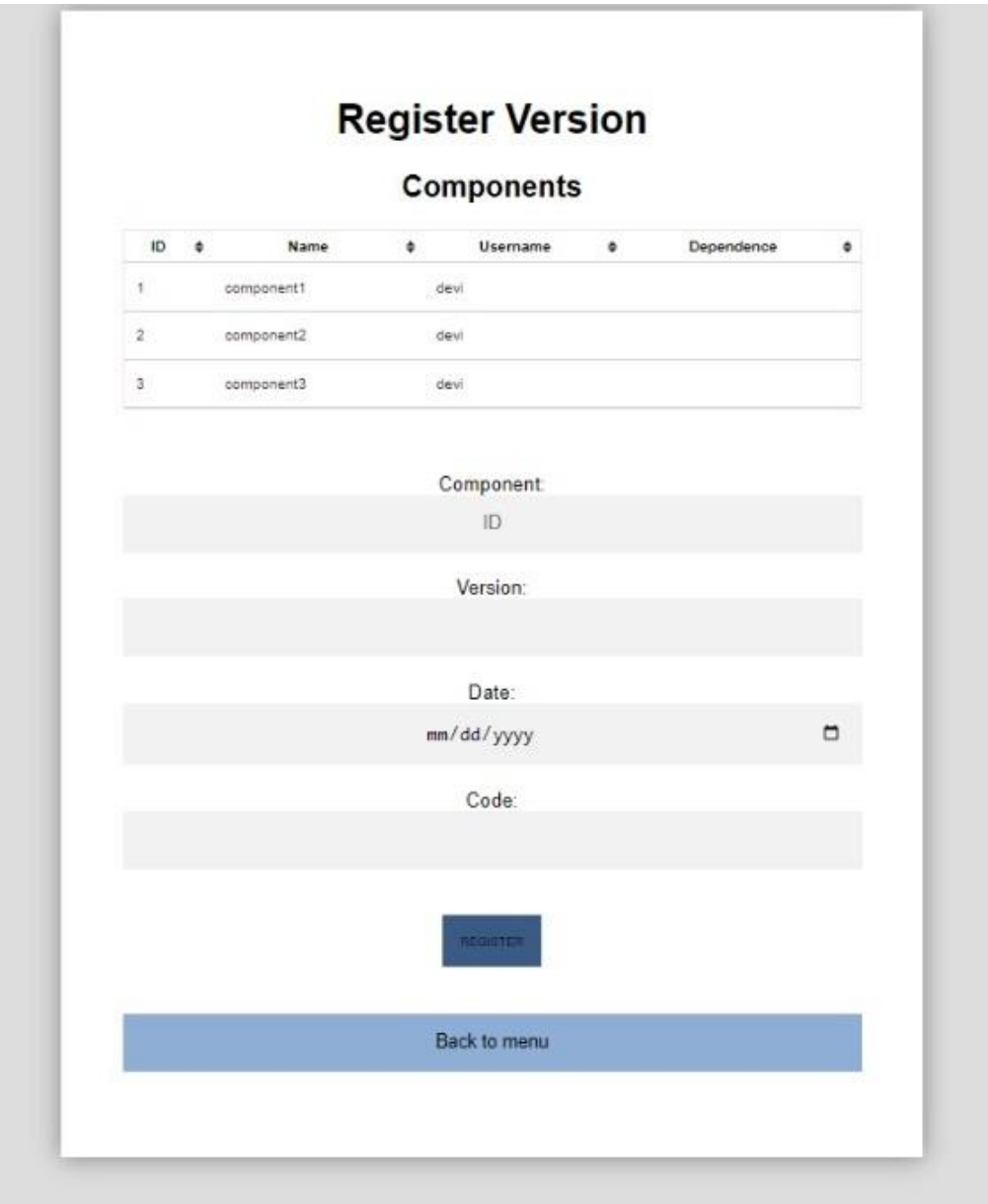
13. Register Component – Component Already Registered



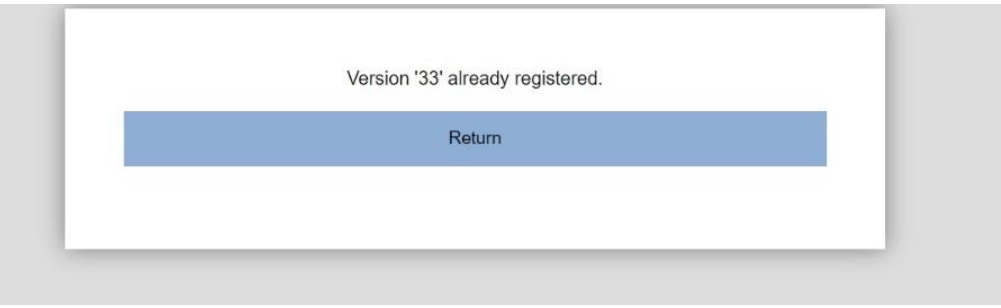
14. Login Form



15. Register Version



16. Register Version – Version Already Registered



WEB TECHNOLOGIES LABORATORY

1. CONNECTION TO THE DATABASE

To set up the database, we used the **MySQL** server, which is managed through **phpMyAdmin** on the **XAMPP** platform. The PHP application connects to the MySQL server using the credentials of an administrator user ('root') and the database connection is established through the **'mysqli_connect'** function. This function accepts the server and user parameters, and then selects the specific database ('srsWeb') using **'mysqli_select_db'**. If the connection or selection fails, the script execution is stopped to ensure that the application does not proceed without a valid connection.

For the data management system, we implemented several tables in the database. Each table was designed based on the provided and developed diagrams. Differently from what had been suggested, we opted to consolidate the admin and user tables into a single 'user' table and introduced a 'userRole', to accommodate potential future user types beyond these two categories. This approach enhances the system's flexibility and scalability to accommodate diverse user roles as needed in the future.

Here is a list of our tables:

The *srsUserRole'* table, as mentioned, was created to define the different roles that users can have within the system, such as administrator or developer. This table contains columns for the role ID and role name, with the ID being the primary key.

The *'srsUser'* table stores user information, including username, password, full name, address, NIF (tax identification number), email, and the user role ID, which is a foreign key referencing the *'srsUserRole'* table. This relationship ensures that each user has an associated role, essential for defining permissions and access within the system.

To manage the system components, the *'srsComp'* table was created. This table records information about each, such as name, description, date, code, creator user ID (foreign key referencing *'srsUser'*), and dependencies between components.

The *'srsCompVersion'* table complements the components table by storing information about the different component versions. This table includes columns for version ID, name, date, code, and component ID (foreign key referencing *'srsComp'*), facilitating version control and updates.

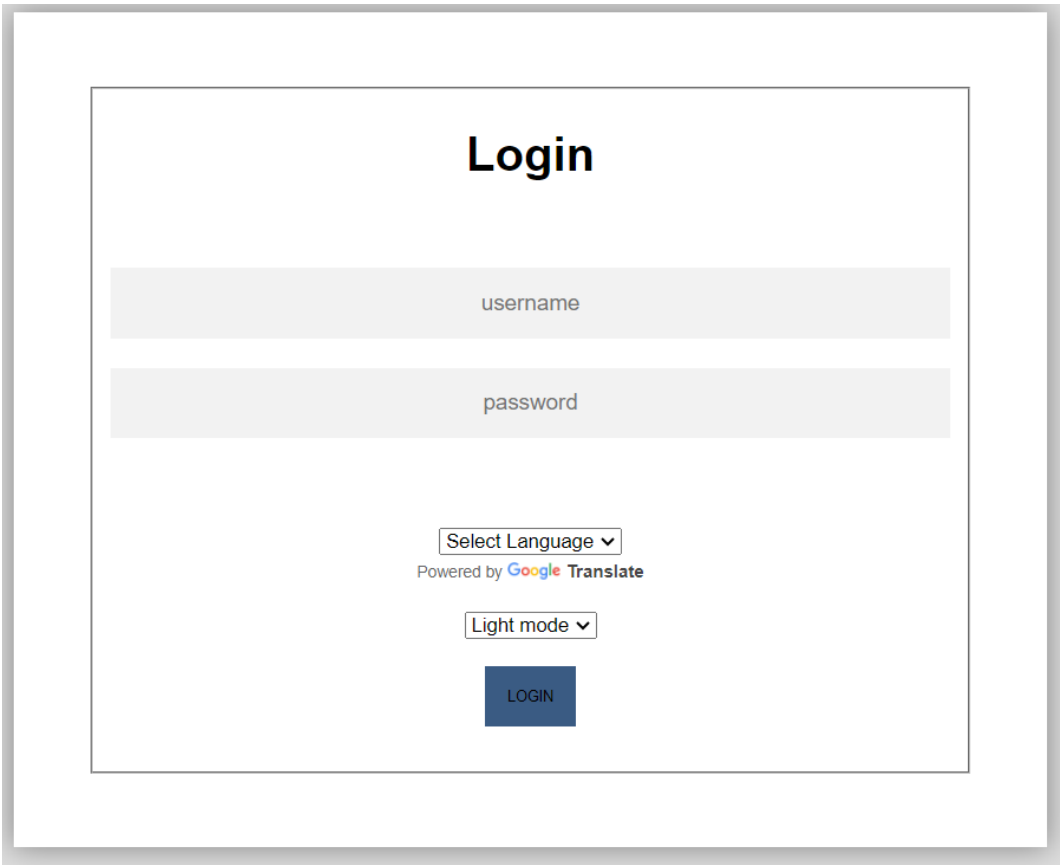
For permission management, the *'srsPerm'* table was implemented to record the different permissions available in the system. The *'srsDevPerm'* table associates these permissions with developers, using foreign keys to link developers (referencing *'srsUser'*) and permissions (referencing *'srsPerm'*).

Finally, the *'srsBackup'* table was created to store backup files, with details about each file, such as name, type, size, URL, upload date, and status. This table is essential for data recovery and system security.

These interconnected tables were designed to maintain data integrity and facilitate system administration.

2. LOGIN SYSTEM

The login page, developed in HTML, provides users with a simple and intuitive interface to input their credentials.



The HTML form collects the username and password, which are then sent via a POST request to the 'login.php' script. PHP sessions, initiated with 'session_start()' on the login page, ensure that any previous session is terminated to prevent session conflicts.

Depending on the user's role retrieved from the database, the script redirects the authenticated user to the appropriate page. Administrators are directed to an administration page ('adminUser.html'), while developers are redirected to a development page ('devUser.html'). Unrecognized users or those with unspecified roles are redirected back to the login page.

3. LANGUAGE AND THEME

To enhance accessibility and user experience customization, we decided to include language selection through Google Translate and theme selection.

The implementation of these features involved the use of JavaScript.

The **'document.addEventListener("DOMContentLoaded", function() { ... })'** function was used to ensure that the code within it is executed only when all the content of the page is fully loaded. This approach prevents any premature manipulations in the Document Object Model (DOM) and ensures a smooth user experience.

The code leverages the browser's local storage to both save and retrieve user preferences. For instance, the user's choice of theme (light or dark) is stored locally. Upon page load, the code checks for a previously selected theme and automatically applies it. Both the functions **'theme()'** and **'saveLanguage()'** function dynamically modify the page based on the user's selection and are responsible for storing the preference selected by the user in the local storage.

Furthermore, our code includes the initialization of the Google Translate translation widget. The **'googleTranslateElementInit()'** function is called to set up and customize the translation widget, allowing for the configuration of available languages and the default page language.

4. SYSTEM FUNCTIONALITIES

Two HTML pages were created for the main user interface: adminUser and devUser. These pages were specifically designed for system administrators and developers to manage the system and each one has dedicated sections tailored to their specific use cases.

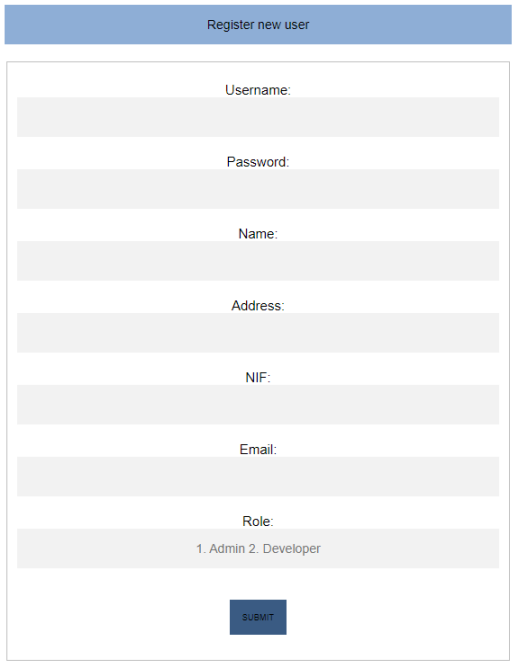
At the beginning of each page, a PHP script checks the user's session to ensure that only authenticated users with the appropriate role can access the page functionalities. Once authenticated, the user is welcomed with a personalized message that displays their username.

```
<?php
    session_start();
    if ($_SESSION["RoleName"] != "admin")
        header("Location: Login/login.html");
?>
<h1>Administrator</h1>
<?php
    echo "<h2>Welcome, " .$_SESSION["UserName"] . "</h2>";
?>
</br>
```

To keep the interface clean and organized, the forms are initially hidden using CSS properties to set their display to "none", making them invisible. When the respective buttons are clicked, the forms become visible. Additionally, the forms can be hidden again by double-clicking. These functionalities were achieved using the 'onclick' and 'ondoubleclick' events.

5. NEW USER REGISTRATION

The new user registration interface was developed using HTML, CSS, and PHP. The HTML page includes a form for the administrator to input the new user's data such as username, password, name, address, NIF, email, and user role.



When the form is submitted, the data is sent to a PHP script, which checks if the username is registered in the database. If the username already exists, a message is displayed to the administrator. Otherwise, the script inserts the new user's data into the database and refreshes the administrator page. If the user role is "Developer", specific permissions are automatically assigned.

The same approach was implemented in additional registration pages, including permissions management, dependencies management, version control, and similar functionalities.

6. PERMISSIONS MANAGEMENT

Firstly, we ensure that developers always have access to components, versions, and dependencies. However, modifying this information requires specific permissions granted by a system administrator. These permissions granted to developers include the ability to add, modify, and remove dependencies, register new components, and versions. Initially, developers have access to all these functionalities, and the administrator can make additions or removals as needed.

To control the visibility of the permission button, the PHP code uses a conditional structure. Here's a more detailed explanation, using the permission "Create new component" as an example:

The PHP code **'<?php if (UserPermissions($ligacao, $userID, 1)): ?>'** is a conditional structure that verifies if the user has the necessary permissions to create a new component. To do this, it calls the 'UserPermissions' function and passes the database connection ('$ligacao'), the user ID ('$userID'), and the specific permission ID needed to create a new component (in this case, 1).

If the 'UserPermissions' function returns true, it means that the user has the appropriate permissions, and as a result, the code inside the conditional block will be executed. Conversely, if the 'UserPermissions' function returns false, it means that the user does not have the necessary permissions, and as a result, the code inside the conditional block will not be executed, and the button will not be shown on the page.

This approach ensures that only users with the correct permissions can access and use the new component creation functionality, thus maintaining the security and integrity of the system.

7. BACKUP AND RESTORE BACKUP

In the final section of our project, we started by creating a file named db.php to configure the connection with our database. We also modified the max_allowed_packet setting in my.cnf to 512M to allow for the storage of larger amounts of data.

Next, to simplify the backup process, we added a button in the HTML file that would allow users to download and insert the file into the created table. For this purpose, we created a file named 'backup.php', that starts by calling db.php to gain access to the database and then save the file path in a variable called dbfilepath. Subsequently, we used the mysql dump command:

"$command = "/Applications/XAMPP/bin/mysqldump --user={$user} --host={$host} {$database} --result-file={$dbfilepath} 2>&1";"

This command dumps the database to the file, which is saved in the previously defined path, and executes it using the exec command.

In the restore backup stage, we began by creating an options menu on the HTML page containing the names of the files saved in the database table. In the PHP file, we, once again, started by calling db.php to define our connection to the database and handle potential errors that may occur during this process.

We created a function that could return false if the data was not found or return the backup data if it existed. The data was then stored in a temporary file using the command: **"file_put_contents($restore_file, $backupData);"**

Finally, we executed the mysql command to restore the selected database.

8. COMPLEMENTARY FEATURE

As an added feature, we've integrated the jQuery plugin TableSorter, allowing users to easily sort table data by clicking on the headers. This enables dynamic sorting of columns in ascending or descending order, providing efficient data organization.

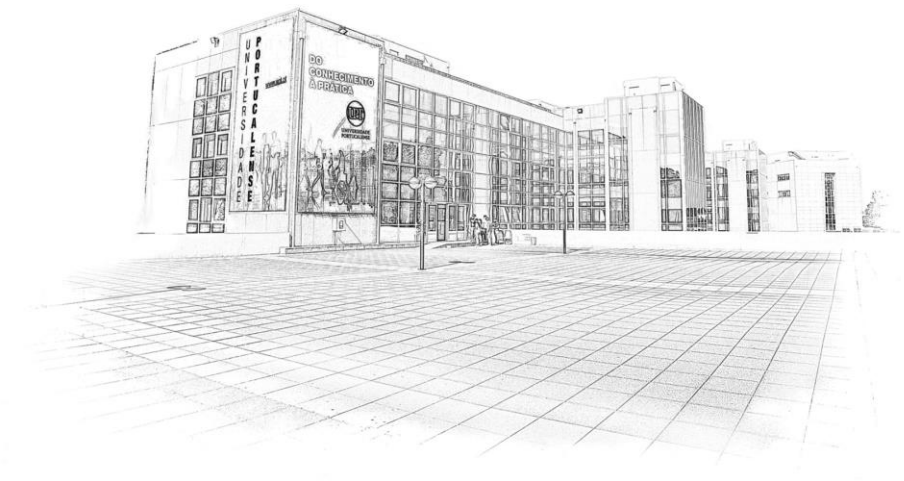
CONCLUSION

This project allowed us to develop essential knowledge for understanding, implementing, and managing software systems, as well as providing a connection between the concepts covered in the two course units.

Throughout the project, we faced various challenges in applying the knowledge acquired in the Software Engineering and Web Technologies Laboratory courses to solve a concrete case study: the Software Component Management System. The project was structured into specific objectives, from creating design specifications to building a functional web application.

In the Software Engineering course, we used modeling tools like Visual Paradigm to create package diagrams, class diagrams, hybrid diagrams, and later an entity-relationship diagram. In the Web Technologies Laboratory course, we developed a database to store system information and created an interactive and secure web application with features like user registration, login/logout, and specific operations based on user type, using programming languages such as JavaScript, PHP, CSS, and HTML.

Despite the challenges, the results achieved reflect our commitment and the application of the knowledge acquired in the classes.



Universidade Portucalense Infante D. Henrique | Rua Dr. António Bernardino de Almeida, 541
4200-072 Porto | Telefone: +351
225 572 000 | email: upt@upt.pt