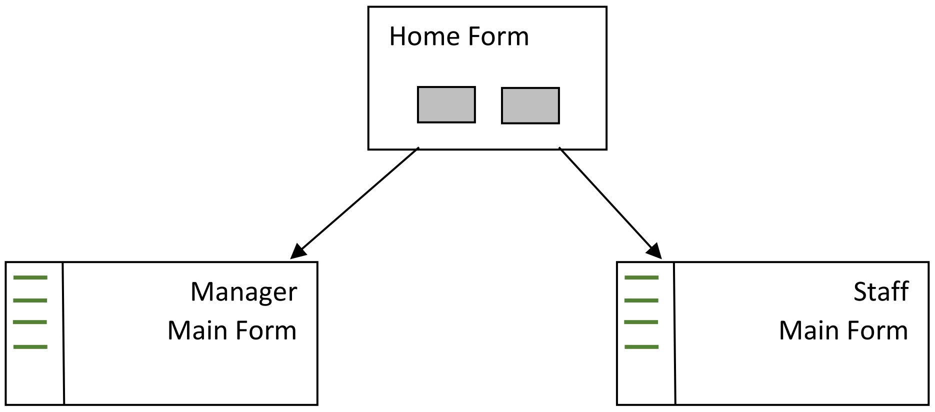
The following document provides a SUGGESTION of how each page COULD look and provides a suggestion of what dependencies the controller would have and how it would interact with any use cases. It does not discuss what the use cases do OR what the repositories would do – you can look at the case study and ask the tutor questions regarding this once implemented.

From the first activities document:



2. Manager main menu	8. Staff main menu
3. View All Staff	9. View User Details
4. Add/Edit Staff	10. Edit User Details
5. View Skills (incl. Allocate a Skill)	5. View Skills (incl. Allocate a Skill)
6. Add/Edit/Delete A Skill	

Contents

Note: that the selection of screens included here reflect some of the use cases identified in the assignment brief – full list below:

Manager

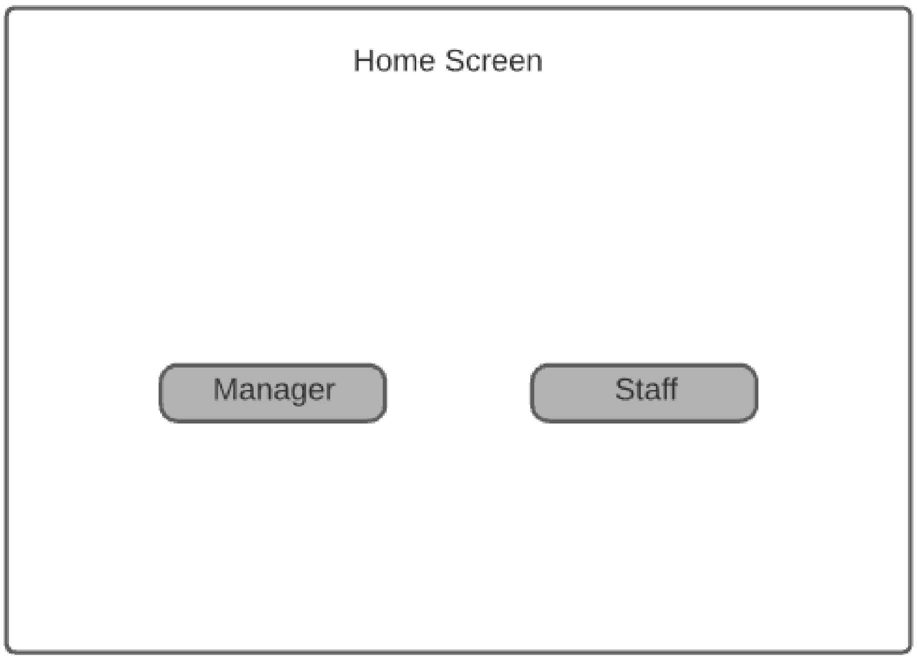
- View all staff working for that manager
  could include a filter by role e.g. all staff that are senior developers
  could filter by particular skills – including level of skill – e.g. all staff with advanced knowledge of Java Programming
- View all skills that can be allocated to a staff member
- Add new staff
- Add new skills
- Edit staff details
- Edit skill information

Staff member

- Allocate/remove a skill to/from yourself
- View skills allocate to you
- View your own personal details
- Edit your own personal details

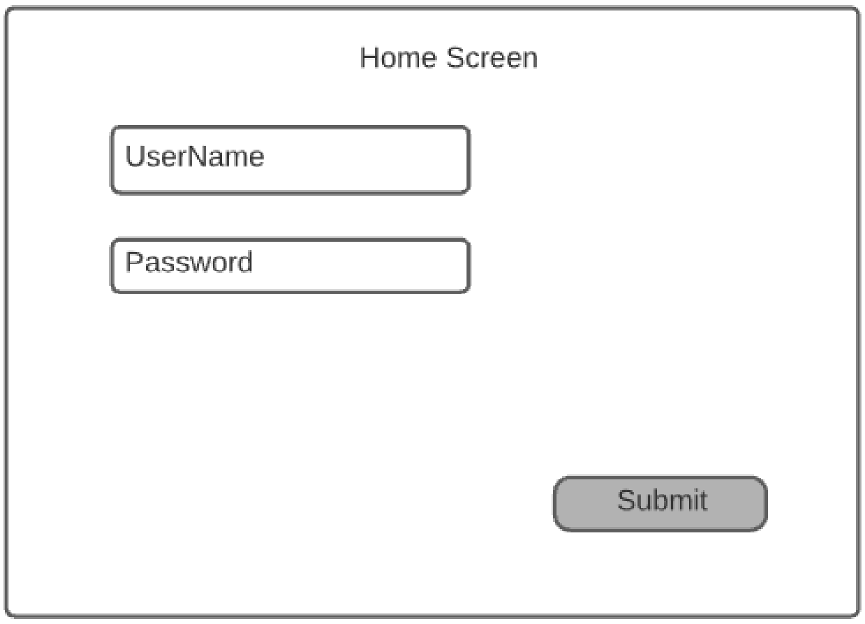
Mark scheme states for 70+ in the implementation section "Code is of an excellent standard and shows **evidence of independent research and application**"
There is an expectation that you will go beyond the functionality and use cases of the assignment to enter the 70+ mark criteria.

1. Mock Up – Home Screen (without login)



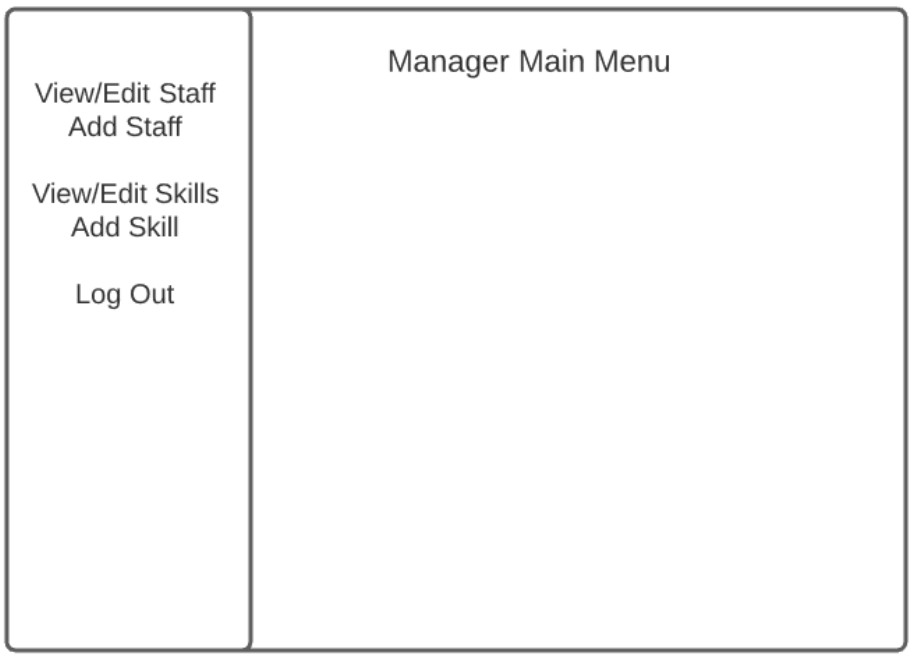
No login so buttons to take you to either the Manager main menu OR Staff main menu pages.

1B. Mock Up – Home Screen (if you had a login – not necessary for assignment 1)



2. Manager - Main Menu

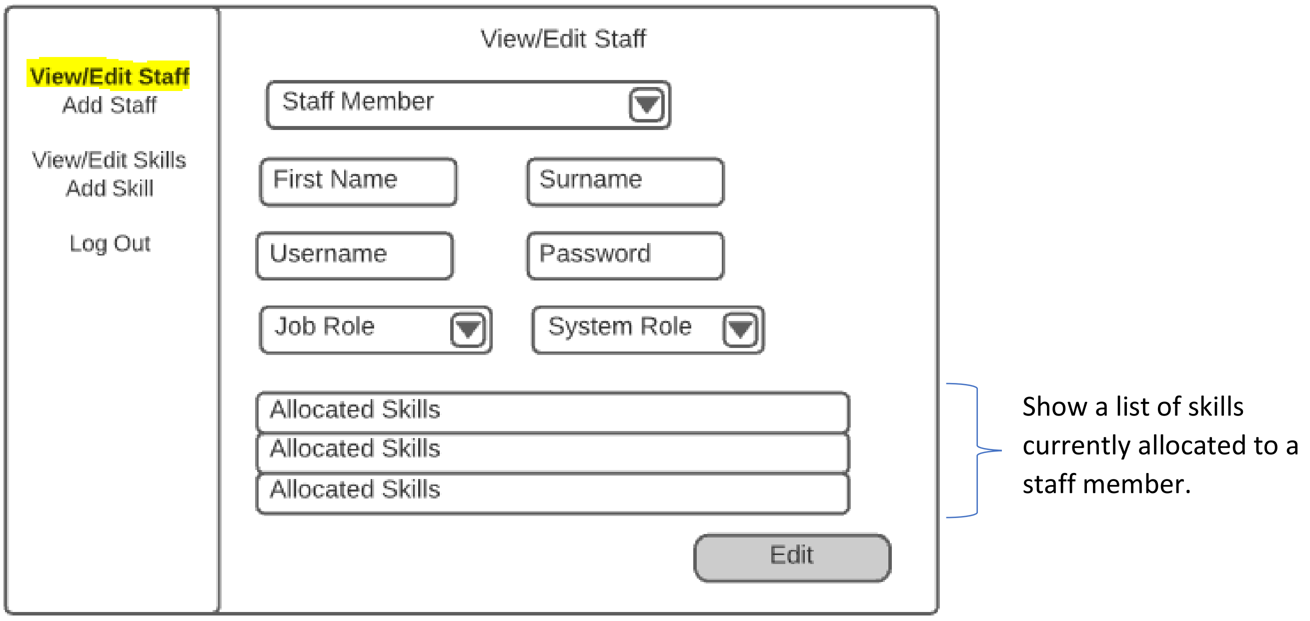
Display main page with nav bar showing all manager links.



Log out here takes you back to the home screen.

3. Manager – View/Edit Staff

Version shown below doesn't include any filters by role or skill (see comment on page 2 – in green).



Controller behaviour

- Dependency on the use case to get all staff.
- Dependency on the use case to edit a staff member.

Initialise method

- Will display a list of staff on loading – here shown as a combo box
- Will select the first staff member
- Will display a list of job roles – here shown as a combo box – with the staff members job role selected.
- Will display a list of job roles – here shown as a combo box – with the staff members job role selected.

How to select an item from a list (or combo) – assuming your list/combo is called myList and you have a method in your staffMember object called getJobRole that returns the JobRole object that represents the staff member's role.

```
myList.getSelectionModel().select(staffMember.getJobRole());
```

When the staff list is clicked

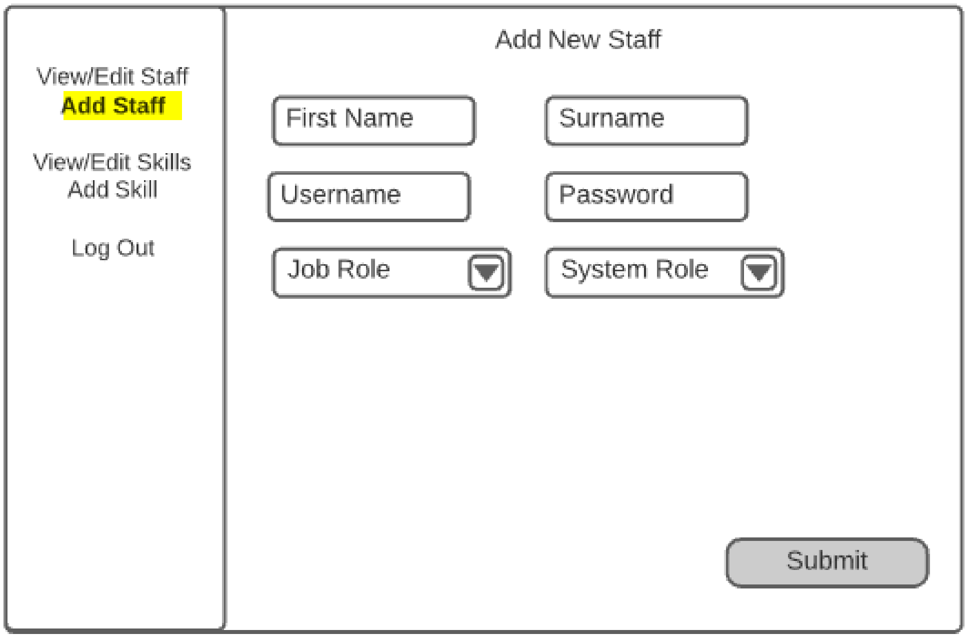
- Will display the skills for that staff member (this will change as you select a different staff member).
  E.g (from the case study – enrol student on module – it details this process from the layout to calling and displaying data when it is clicked on
  `<ListView fx:id="someList" onMouseClicked="#someMethodToCall"/>`

Edit – in a try block - catch IllegalArgumentException (see Edit Module an example)

- Assign the StaffMember to a local StaffMember variable.

- Assign first name, surname, username and password to local string variables in the add New Staff method.
- Assign the Job role from the job role list to a local job role variable.
- Assign the Job role from the job role list to a local system role variable.
- Pass all of these values to your edit a staff member use case individually using requestList.add then call its execute method.

4. Manager – Add New Staff



Skills will be allocated by the staff member when they log in – not by the administrator (for simplicity).

Controller behaviour

- Dependency on the use case to add new staff.

Initialise method

- Call enum to get all job roles and display in the job role list. Select the first.
- Call enum to get all system roles and display in the system role list. Select the first

Reminder of how to populate a list from Enums:

```
@FXML

private ListView<RouteNames> routesList;

public void initialize() {
    ObservableList<RouteNames> items =
FXCollections.observableArrayList(RouteNames.values());
    routesList.setItems(items);
    routesList.getSelectionModel().selectFirst();
}
```

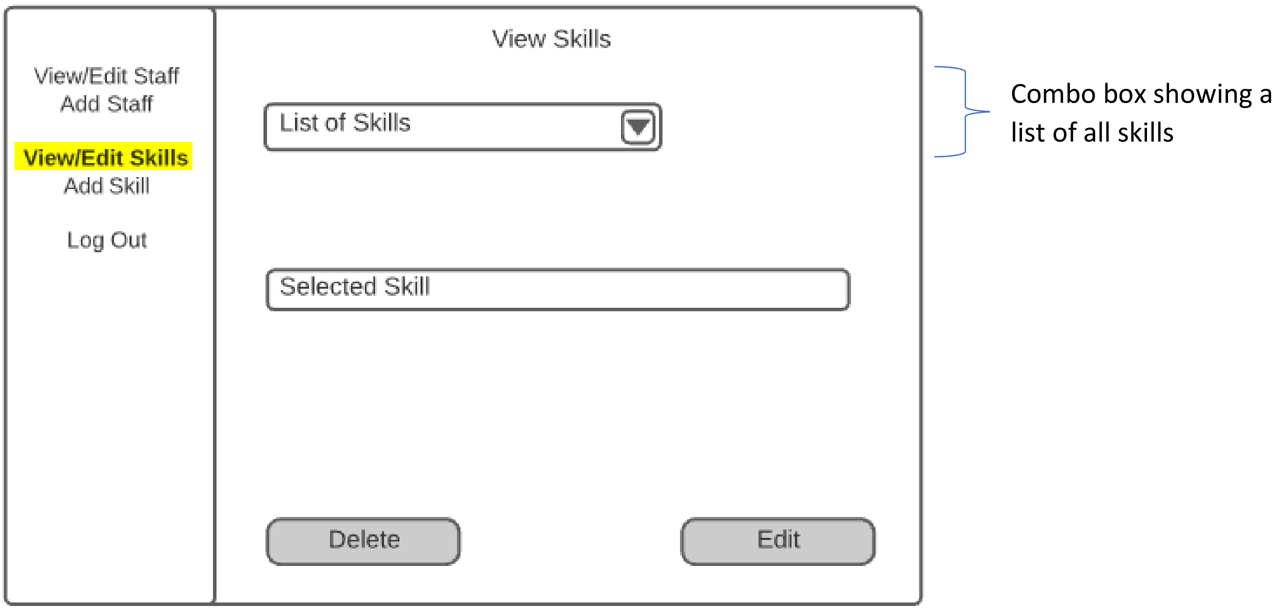
Submit - (method might be called addNewStaff)

In a try block

- Assign first name, surname, username and password to local string variables in the add New Staff method.
- Assign the Job role from the job role list to a local job role variable.
- Assign the System role from the job role list to a local system role variable.
- Pass all of these values to your use case individually using requestList.add then call its execute method.

5. Manager – View/Edit Skills

Allow the manager to see all skills and then select a skill that they can then either delete or edit.



Controller behaviour

- Dependency on the use case get all skills.

Initialise method

- Will display a list of skills on loading – here shown as a combo box
- Will select the first skill in the list

When the skill list is clicked

- Will display the skill name for that skill – so that we can edit it if we wish.

Delete (look at handleWithdrawStudent in the EnrolStudentOnModuleController for an example)

- Assign the Skill from the list to a local Skill variable.
- Pass this skill variable to the delete use case using requestList.add then call its execute method

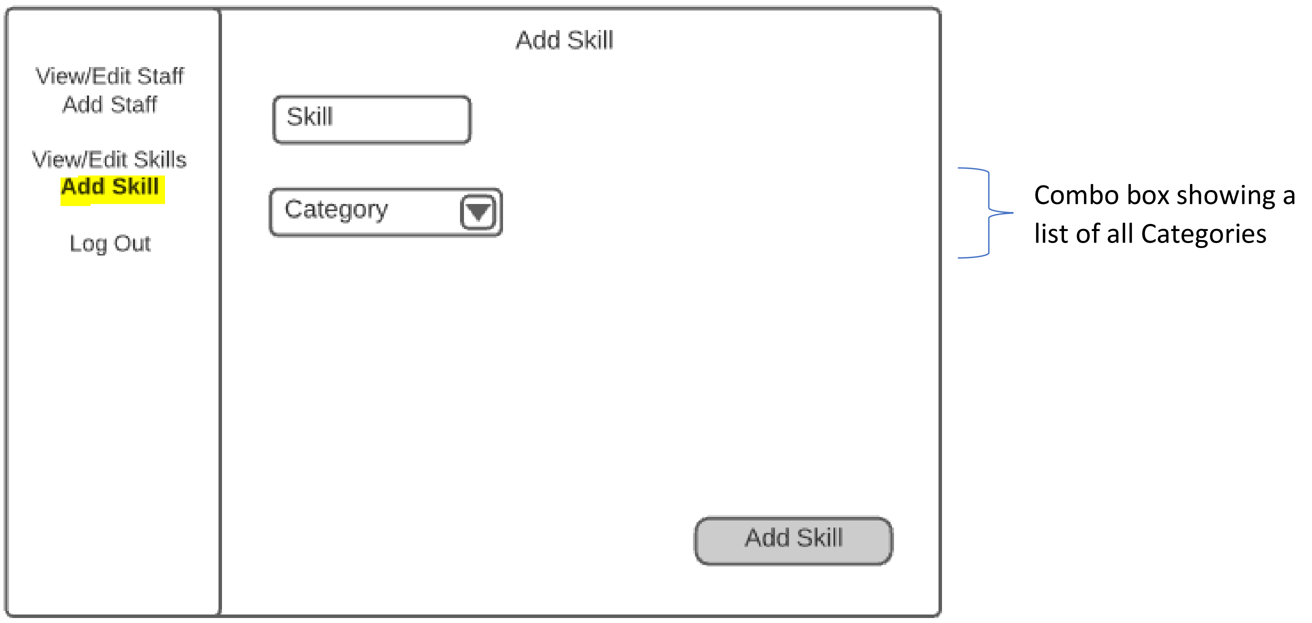
Edit

In a try block – catch IllegalArgumentException (see Edit Module an example)

- Assign the Skill from the list to a local Skill variable.
- Pass the selected skill variable to the edit use case using requestList.add then call its execute method

6. Manager – Add (New) Skill

Adding a new skill to the database (not allocating to a user).



Controller behaviour

- Dependency on the use case to get all categories.
- Dependency on the use case to add a new skill.

Initialise method

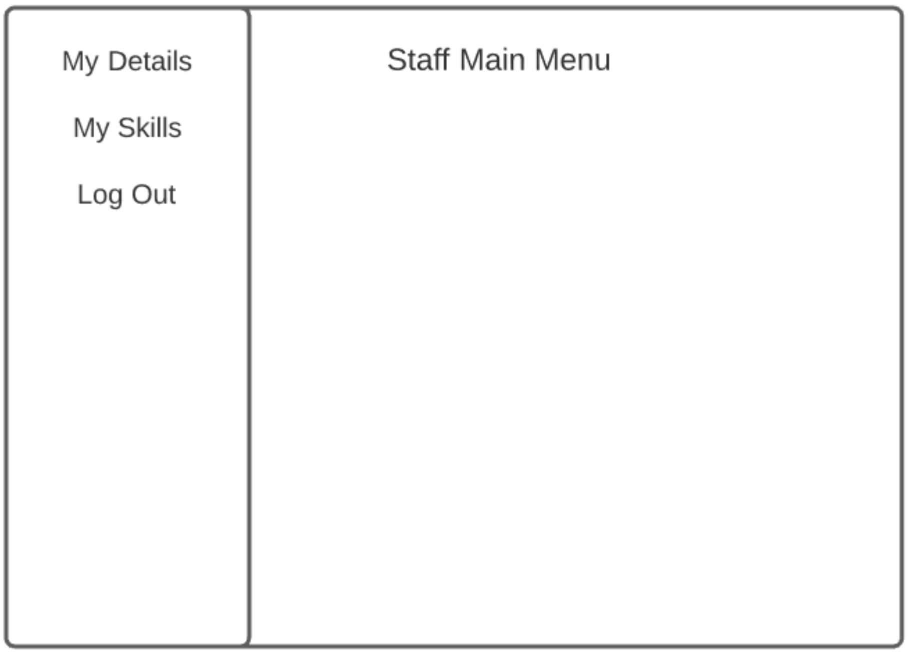
- Will display a list of categories on loading – here shown as a combo box
- Will select the first category in the list

Add

- Assign the Skill name from the text box to a local String variable.
- Pass the selected category variable to the edit use case using requestList.add then call its execute method

7. Staff - Main Menu

Display main page with nav bar showing all manager links.



Log out here takes you back to the home screen.

Question: If we don't log in as such how does the system 'know' which user details to display – in each of the StaffUser pages – shown on the following pages?

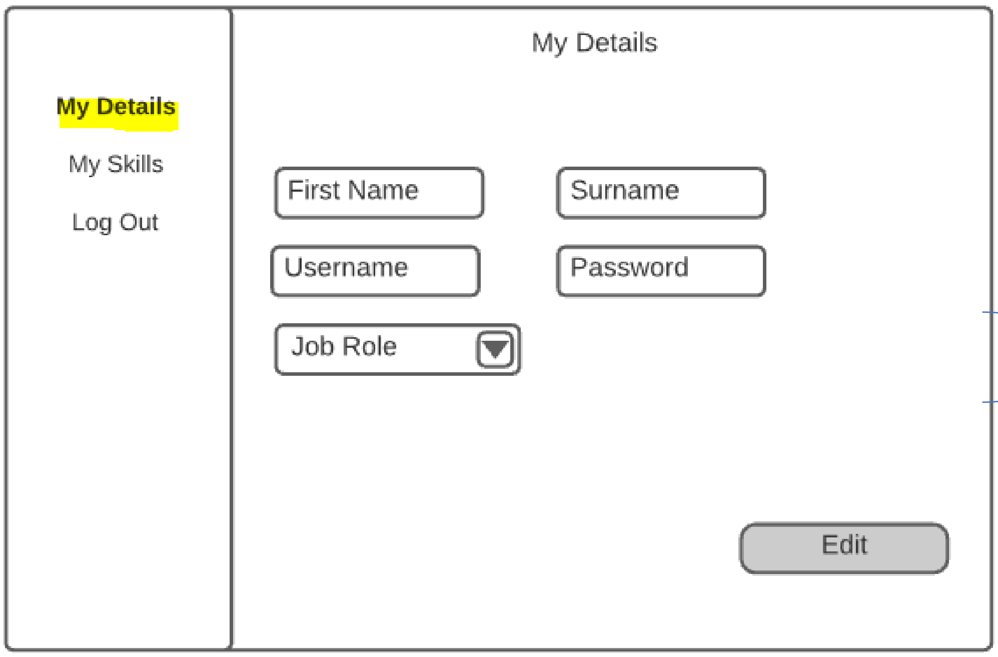
One solution is to get the first StaffUser (or whatever your class is called) object from the Ioc_Container class in the controller where it is needed and pass this to the relevant use case via requestList.add.

The following would retrieve the first StaffUser object from a getStaffUserRepository method in the Ioc_Container:

```
e.g. StaffUser loggedInUser = Ioc_Container.getStaffUserRepository().getAll().get(0);
     myUseCase.requestedList.add(loggedInUser);
```

8. Staff – View/Edit My Details

Display the staff user details. Enable these to be edited.



Combo box showing a list of job roles

Controller behaviour

Dependency on the use case to edit the user details.

Initialise method

- Will display a list of categories on loading – here shown as a combo box
- Will select the first category in the list

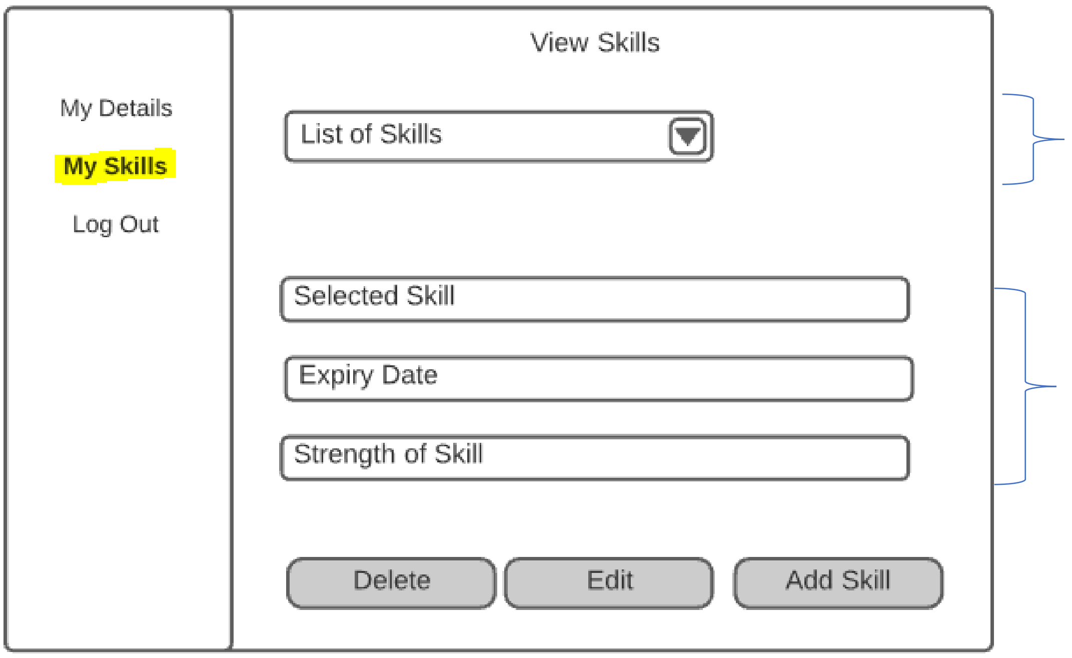
Edit

- Assign first name, surname, username and password to local string variables in the add New Staff method.
- Assign the Job role from the job role list to a local job role variable.
- Pass all of these values to your use case individually using requestList.add then call its execute method.

9. Staff – View/Edit My Skills

View all skills allocated to the staff member (user skills) – when you select one you can see that skill below with its expiry strength of skill – and a notes text box (not shown).

Note: It is also possible to design a list view with all the skills and have the toString showing the expiry (if there is one) and then have a notes box that could display notes for that particular user skill (if you have added anything).



Combo box showing a list of all skills allocated to the staff member

Details of selected user skill.

Expiry date can use a date picker (see the case study enrolment).

Strength of skill is a list or combo.

Controller behaviour

- Dependency on the use case get all skills.
- Dependency on the use case delete a skill.
- Dependency on the use case add a skill.
- Dependency on the use case edit a skill.

Initialise method

- Will display a list of skills on loading – here shown as a combo box
- Will select the first skill in the list
- Populate the strength of skill list/combo from the enum

When the skill list is clicked

- Will display the user skill details for the selected skill.

Delete

- Assign the Skill from the list to a local Skill variable.
- Pass this skill variable to the delete use case using requestList.add then call the delete use case execute method

Edit

- Assign the Skill from the list to a local Skill variable.

- Pass this skill variable to the edit use case using requestList.add
- Get the value of the date picker (for expiry) and the pass this skill variable to the edit use case using requestList.add
- Get the selected strength of skill from the combo/list and pass this skill variable to the edit use case using requestList.add
- Call the edit use case execute method

Add

- Assign the Skill from the list to a local Skill variable.
- Pass this skill variable to the edit use case using requestList.add
- Get the value of the date picker (for expiry) and the pass this skill variable to the edit use case using requestList.add
- Get the selected strength of skill from the combo/list and pass this skill variable to the edit use case using requestList.add
- Call the add use case execute method