Software Requirements Specification

for

POAM

Version 1.4

Prepared by Pop Diana-Stefania and Gherghina Andrei-Cornel

Faculty of Automation, Computers and Electronics

March 2019

Table of Contents

Table of Contents ii

Revision History iii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Product Scope 1

1.5 References 1

2. Overall Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 3

2.5 Design and Implementation Constraints 3

2.6 User Documentation 4

2.7 Assumptions and Dependencies 4

3. External Interface Requirements 4

3.1 User Interfaces 4

3.2 Hardware Interfaces 16

3.3 Software Interfaces 16

3.4 Communications Interfaces 18

4. System Features 18

4.1 Log in 18

4.2 Main view 18

4.3 Employees list 19

4.4 Edit employee 19

4.5 Add employee 20

4.6 Owners list 20

4.7 Edit owner 21

4.8 Add owner 21

4.9 Apartment details 21

4.10 Apartments owned list 22

4.11 Add payment receipt 22

4.12 Add water consumption 23

4.13 Add apartment 23

4.14 Contracts list 23

4.15 Add contract 24

4.16 Contract details 24

5. Other Nonfunctional Requirements 25

5.1 Performance Requirements 25

5.2 Safety Requirements 25

5.3 Security Requirements 25

5.4 Software Quality Attributes 25

5.5 Business Rules 25

6. Other Requirements 26

Appendix A: Glossary 26

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Gherghina Andrei | 07.03.2019 | Initial form | 1.0 |
| Pop Diana | 07.03.2019 | Edit several document sections | 1.0.1 |
| Pop Diana | 09.03.2019 | Added sections 2.4. Operating Environment and 2.5. Design and Implementation Constraints | 1.0.2 |
| Gherghina Andrei | 09.03.2019 | Added section 5.4 Software Quality Attributes | 1.0.3 |
| Pop Diana | 09.03.2019 | Edit sections 2.2. Product Functions and 2.3. User classes and Characteristics | 1.0.4 |
| Gherghina Andrei | 10.03.2019 | Added section 3.1. User Interfaces | 1.1 |
| Pop Diana | 10.03.2019 | Added sections 4. System Features and Appendix A: Glossary | 1.2 |
| Gherghina Andrei | 11.03.2019 | Edit section 3.1. User Interfaces | 1.2.1 |
| Pop Diana | 11.03.2019 | Added section 3.3. Software Interfaces | 1.3 |
| Gherghina Andrei | 13.03.2019 | Minor edits to sections 1.4. Product Scope, 5.3. Security Requirements and 5.1. Performance Requirements | 1.3.1 |
| Pop Diana | 13.03.2019 | Minor edits to sections 1.5. References and 3.3. Software Interfaces | 1.4 |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

The purpose of this document is to present a detailed description of the software POAM. It will explain the purpose and features of the software, its interfaces, what the software will do and the constraints under which it must operate. This document is intended for users and also potential developers.

POAM (Property Owners Association Manager) is a web application used to manage the activities of an owners’ association.

The product scope of the application is to help users manage the contracts with the suppliers, the employees and payroll management, and the management of apartments in the association.

## Document Conventions

This Document was created based on the IEEE template for System Requirement Specification Documents.

For additional abbreviations and acronyms translation see Appendix A.

## Intended Audience and Reading Suggestions

• Typical users (property owners and administrators) who want to use POAM.

• Programmers who are interested in working on the project by further developing it or fix existing bugs.

## Product Scope

The product scope of the **POAM** application is to help administrators manage the contracts with the suppliers, the employees and payroll management, and the management of apartments in the association.

The owner is able to add apartments, edit his own account information, add payments receipts and water consumption.

## References

* *ASP.NET Core MVC tutorial:*

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-2.2>

* *Understanding ASP.NET Core 2.x:*

<https://www.pluralsight.com/courses/understanding-aspdotnet-core-2x>

* *Project management tool:*

<https://kanbanflow.com/>

* *Online diagram software:*

<https://www.draw.io/>

# Overall Description

## Product Perspective

POAM (Property Owners Association Manager) is a web application used to manage the activities of an owners’ association.

It is a web-based system implementing client-server model.

The computer system includes a user interface, a database server that will store

information about the activity of the association, an application server. The user has access to the service only through the user interface.

## Product Functions

A short list of main product functions:

* **List Employees** -> View a list of current employees and their specific information.
* **List Owners** -> View a list of all association owners.
* **List Contracts** -> View a list of all existing contracts with providers.
* **Apartments Owned** -> View a list of all apartments owned by a logged user.
* **Add Apartment** -> Add a new apartment to apartments owned list.
* **Add Employee** -> Add a new employee to the employee database.
* Edit existing employee’s information.
* Delete existing employees from the database.
* Edit owner account information.
* Delete owner account.
* **Add Owner** -> Add a new owner account.
* **Add Contract** -> Add new contracts with providers for the current month.
* **Finalize** -> End contracts with providers for the current month.
* **Add payment receipt** -> Add apartment payment receipt for the current month.
* **Add water consumption** ->Adding water consumption for each owned apartment.
* View any apartment’s total payment debts at any time.
* View a report of amounts to be cashed from every owner of the association.

## User Classes and Characteristics

The user classes of this product are:

1. **Administrator (Association administrator)** with rights to:

* View a list of current employees and their specific information
* View a list of all association owners.
* View a list of all existing contracts with providers.
* View a list of all apartments owned.
* Add a new apartment to apartments owned list.
* Add a new employee to the employee database.
* Edit existing employee’s information.
* Delete existing employees from the database.
* Edit own account information.
* Delete owner account.
* Add a new owner account.
* Add new contracts with providers for the current month.
* End contracts with providers for the current month.
* Add apartment payment receipt for the current month.
* Add water consumption for each owned apartment.
* View any apartment’s total payment debts at any time.
* View a report of amounts to be cashed from every owner of the association.

1. **Typical user (Property owner)** with rights to:

* View a list of all association owners.
* View a list of all apartments owned.
* Add a new apartment to apartments owned list.
* Edit own account information.
* Add apartment payment receipt for the current month.
* Adding water consumption for each owned apartment.
* View any apartment’s total payment debts at any time.
* View a report of amounts to be cashed from every owner of the association.

Note: The administrator will also have typical user rights (the association administrators could be owners of their own).

## Operating Environment

**Client:**

* Operating system: Windows 7/8/8.1/10
* Internet browser: Google Chrome 70.0+

**Application server requirements:**

* Database server: Microsoft SQL Server 2017
* Minimum system requirements:

1. Processor: 2 x 1.6 GHz CPU
2. RAM: 3.5 GB
3. Free disk space: 3 GB

## Design and Implementation Constraints

**Developer tools used:**

* Microsoft Visual Studio 2017 IDE (Community or Professional)
* Microsoft SQL Server Express 2017
* Microsoft SQL Server Management Studio 2017

**Framework:** ASP.NET Core MVC 2.1

## User Documentation

This document can be used as user documentation.

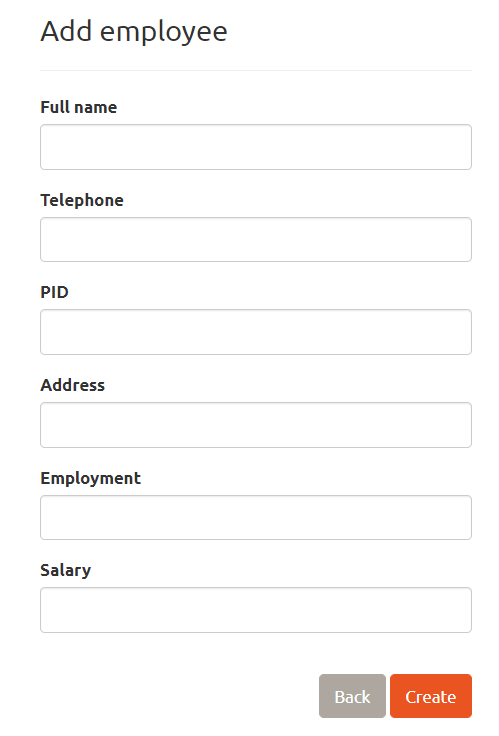
## Assumptions and Dependencies

Not applicable.

# External Interface Requirements

## User Interfaces

1. Add employee:



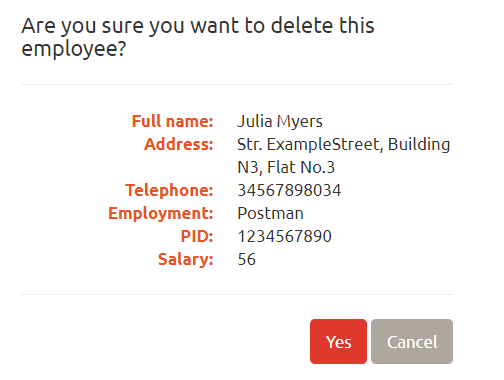
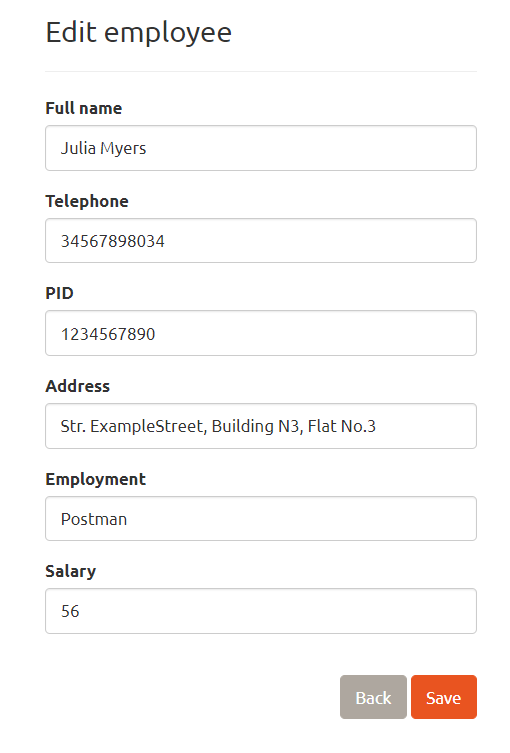
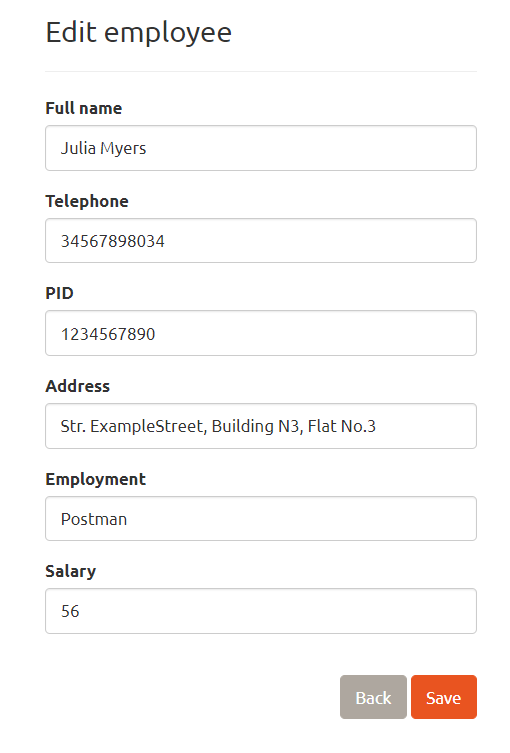
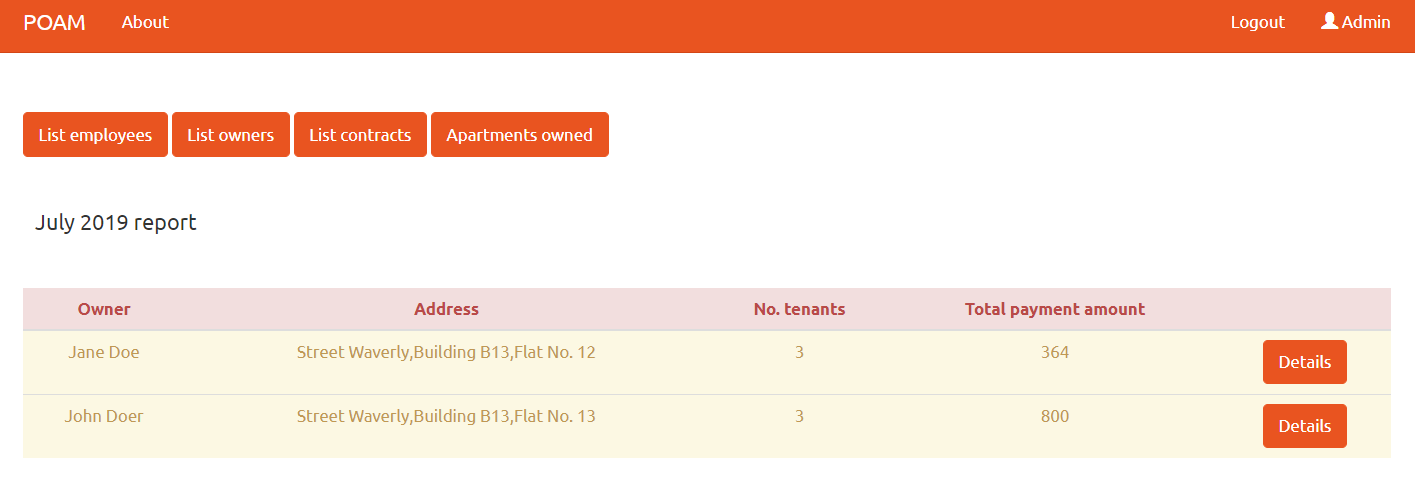
1. Employee deletion confirmation:
2. Edit employee:

Figure 4

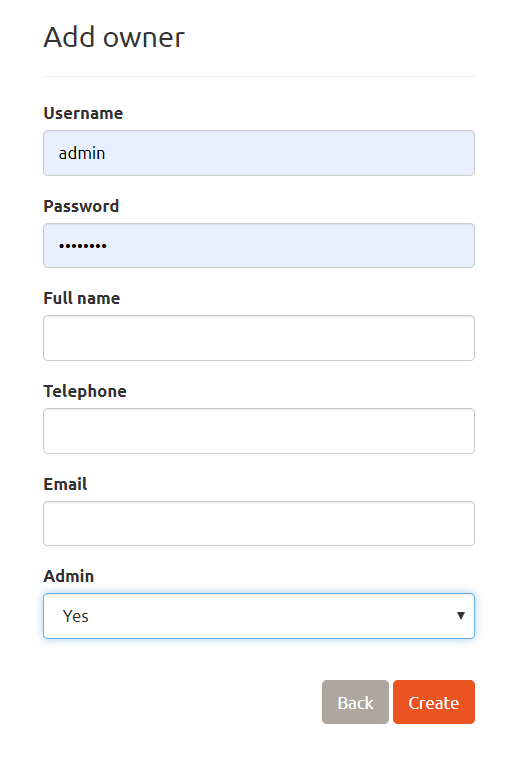




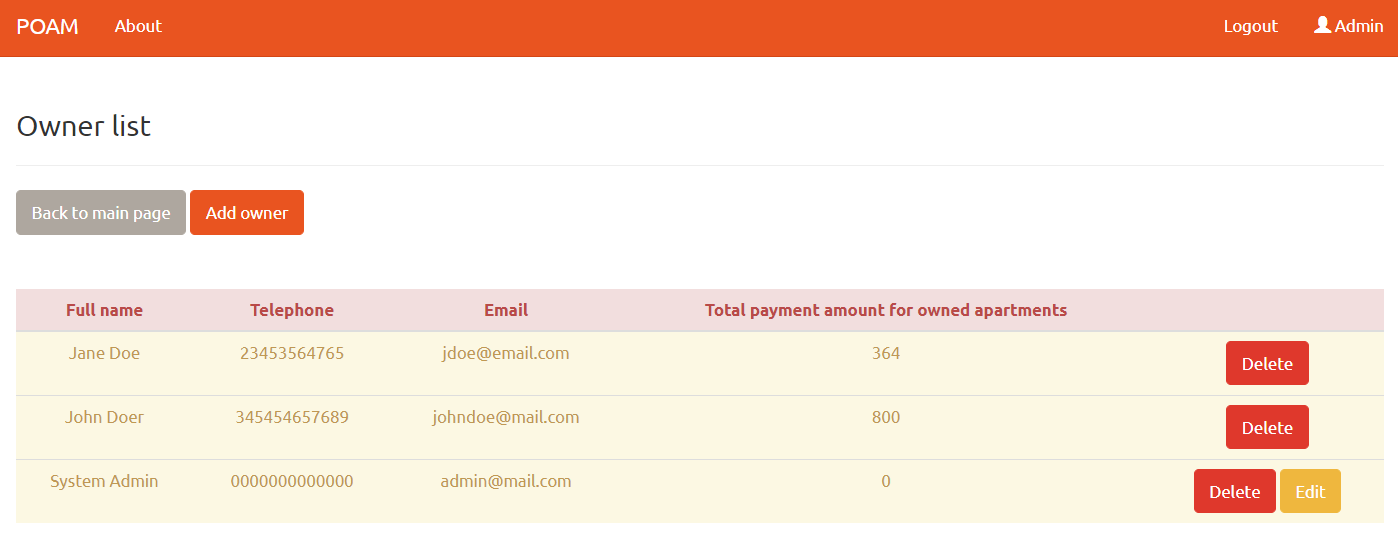
1. Main page (admin view):



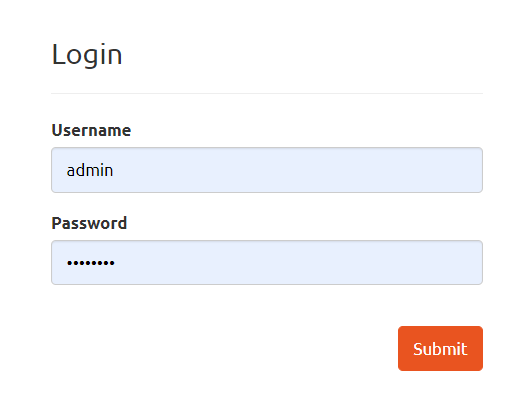
1. Add new owner account:



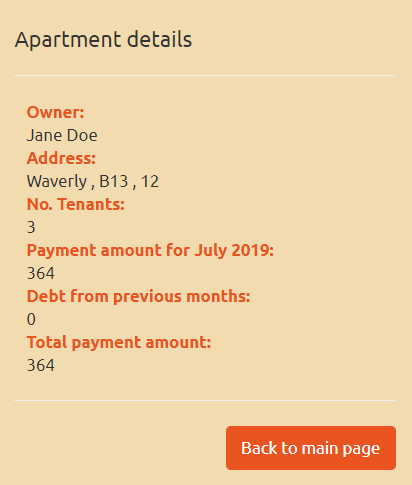
1. Owner list (admin view):



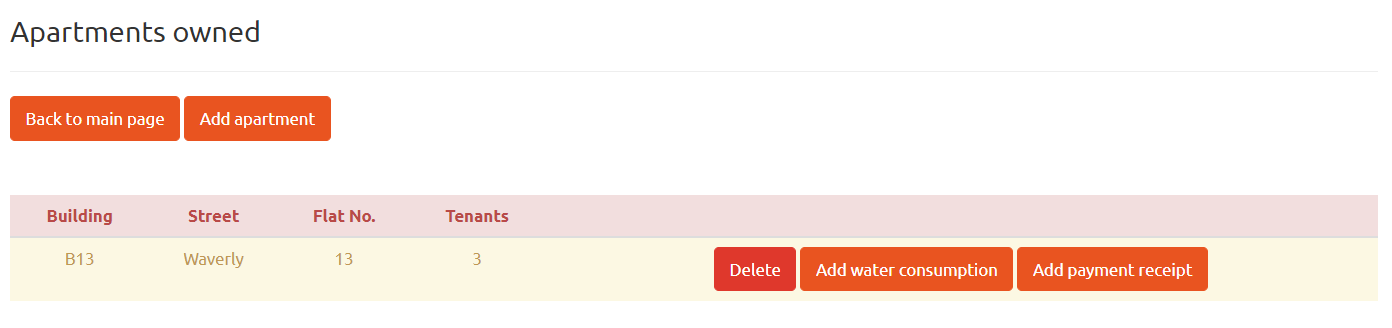
1. Log in:



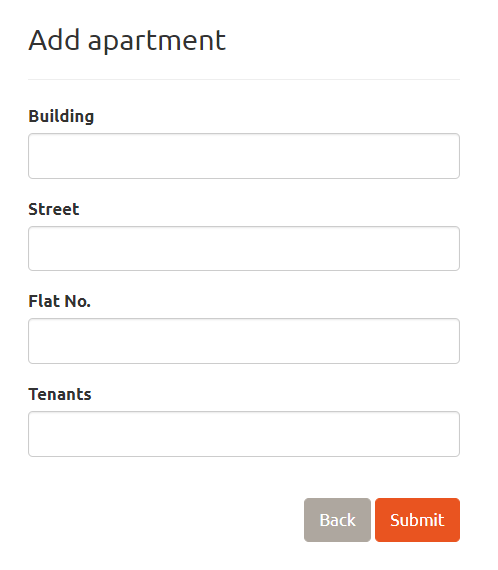
1. Apartment details:



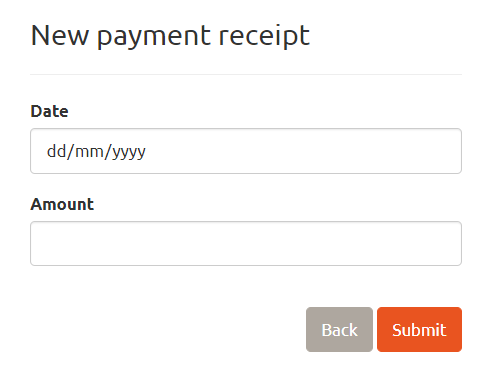
1. Apartments owned:



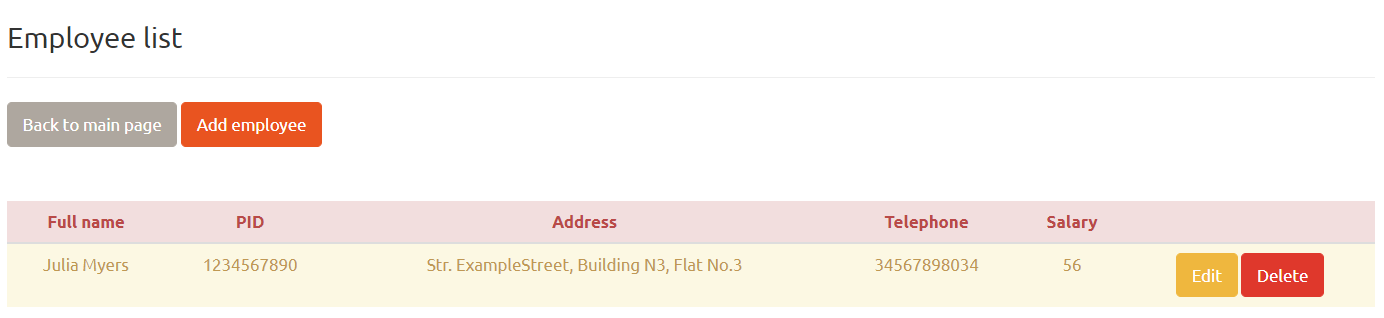
1. Add apartment:



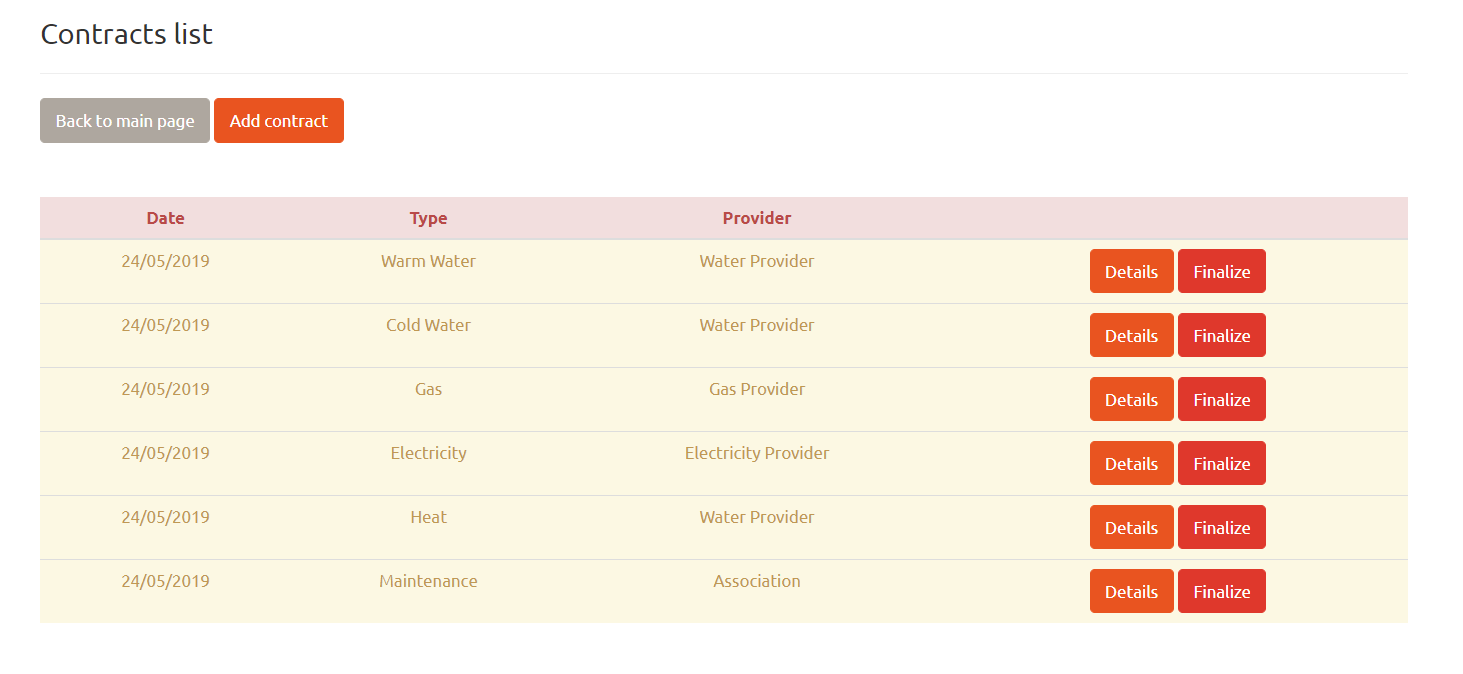
1. New payment receipt:



1. Employees list:



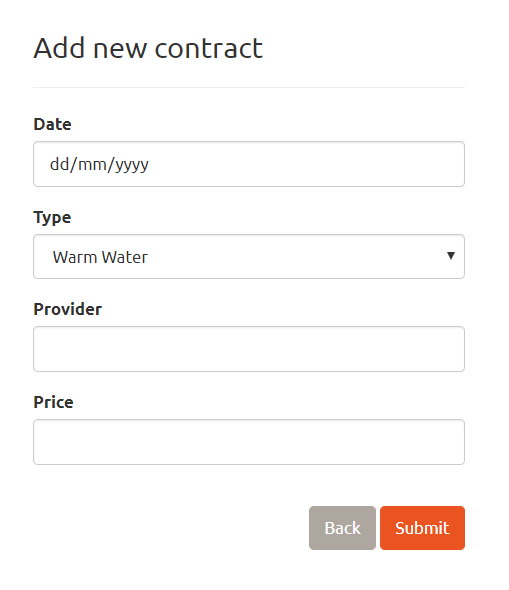
1. Contract list:



1. Contract details:



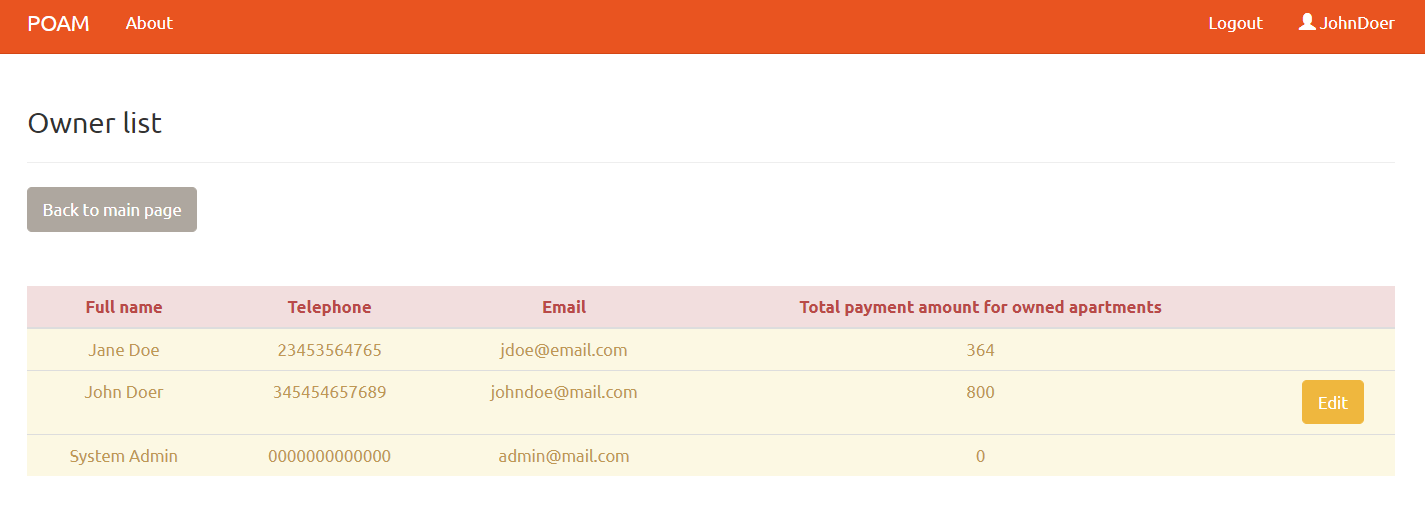
1. Add contract:



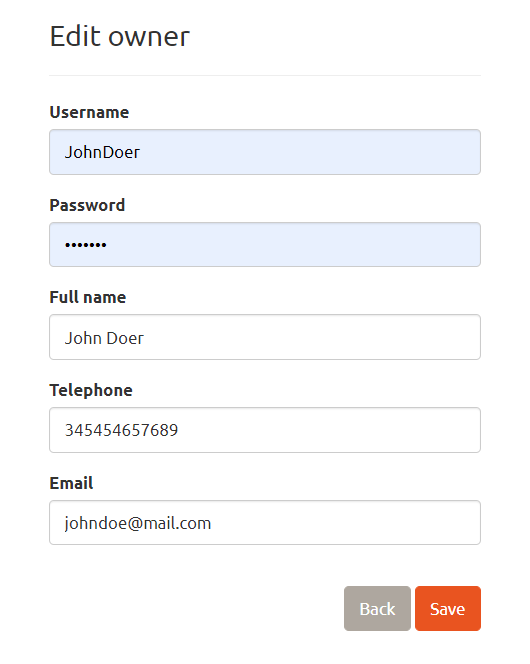
1. Main page (user view):



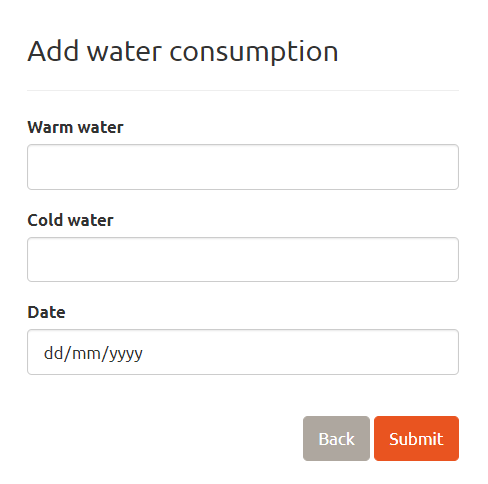
1. Owner list (user view):



1. Edit owner:



1. Add water consumption:



## Hardware Interfaces

Not applicable.

## Software Interfaces

List of all database R/W function prototypes:

* **Login** (User user)

*Retrieves a list of all users’ information from the database in order to login.*

* **MainPage** ()

*Retrieves a list of all apartments’ information from the database.*

* **EmployeesList** ()

*Retrieves a list of all employees’ information from the database.*

* **EditEmployee** (int? id)

*Retrieves a certain employee’s information from the database (by id).*

* **EditEmployee** (int id, Employee employee)

*Updates a certain employee’s information in the database (by id).*

* **DeleteEmployee** (int? id)

*Retrieves a certain employee’s information from the database (by id).*

* **DeleteEmployeeConfirmed** (int id)

*Deletes a certain employee’s information from the database (by id).*

* **AddEmployee** (Employee employee)

*Adds a new employee’s information in the database.*

* **EmployeeExists** (int id)

*Verifies if a certain employee exists in the database (by id).*

* **OwnersList** ()

*Retrieves a list of all owners’ information from the database.*

* **EditOwner** (int? id)

*Retrieves a certain owner’s information from the database (by id).*

* **EditOwner** (int id, Owner owner)

*Updates a certain owner’s information in the database (by id).*

* **AddOwner** (Owner owner)

*Adds a new owner’s information in the database.*

* **DeleteOwner** (int? id)

*Retrieves a certain owner’s information from the database (by id).*

* **DeleteOwnerConfirmed** (int id)

*Deletes a certain owner’s information from the database (by id).*

* **OwnerExists** (int id)

*Verifies if a certain owner exists in the database (by id).*

* **AddPaymentReceipt** (Receipt receipt)

*Add a new payment receipt in the database.*

* **ApartmentDetails** (int? id)

*Retrieves a certain apartment’s information from the database (by id).*

* **AddWaterConsumption** (WaterConsumption consumption)

*Add new water consumption information in the database.*

* **AddApartment** (Apartment apartment)

*Adds a new apartment’s information in the database.*

* **DeleteApartment** (int? id)

*Retrieves a certain apartment’s information from the database (by id).*

* **DeleteApartmentConfirmed** (int id)

*Deletes a certain apartment’s information from the database (by id).*

* **ApartmentExists** (int id)

*Verifies if a certain apartment exists in the database (by id).*

* **ContractList** ()

*Retrieves a list of all contracts’ information from the database.*

* **AddContract** (Contract contract)

*Adds a new contract’s information in the database.*

* **ContractDetails** (int? id)

*Retrieves a certain contract’s information from the database (by id).*

* **FinalizeContract** (int? id)
* **FinalizeContractConfirmed** (int id)

*Deletes a certain contract’s information from the database (by id).*

* **ContractExists** (int id)

*Verifies if a certain contract exists in the database (by id).*

* **ApartmentsOwned** ()

*Retrieves a list of all apartments’ information from the database.*

* **ApartmentsOwned** (int id, Apartment apartment)

*Updates a certain apartment’s information in the database (by id).*

## Communications Interfaces

Not applicable.

# System Features

## Log in

4.1.1 Description and Priority

User (administrator and typical user) authentication. High priority, risk 8 (from 0 to 9) due to the application’s architecture and security.

4.1.2 Stimulus/Response Sequences

Input: User’s username and password, submit button.

Output: Log in confirmation (redirect to main page).

4.1.3 Functional Requirements

REQ-1: Input textbox for Username

REQ-2: Input textbox for Password

REQ-3: Submit button

## Main view

4.2.1 Description and Priority

Provides a report with every apartment’s total payment debts in the current month and access to user functions depending on the type of user logged in. High priority – main page of the application.

4.2.2 Stimulus/Response Sequences

Different views for each type of user logged in.

* Administrator:

Input: List Employees, List Owners, List Contracts, Apartments owned buttons and a details button for each apartment listed in the report.

Output: A list with apartments’ information – owner, address, no. tenants, total payment amount for the current month.

* Typical user:

Input: List Owners, Apartments owned buttons and a details button for each apartment listed in the report.

Output: A list with apartments’ information – owner, address, no. tenants, total payment amount for the current month.

4.2.3 Functional Requirements

REQ-4: List Employees button

REQ-5: List Owners button

REQ-6: List Contracts button

REQ-7: Apartments owned button

REQ-8: A button for each listed apartment’s details (Details)

## Employees list

4.3.1 Description and Priority

Provides a list of all existing employees in the association and CRUD operations with their information. A feature only available to administrators.

Medium priority.

4.3.2 Stimulus/Response Sequences

Input: A button that redirects to the main page, a button to add a new employee to the database and buttons for every employee’s details, deletion, and editing.

Output: A list with employees’ information – full name, telephone, salary.

4.3.3 Functional Requirements

REQ-9: Back to Main Page button

REQ-10: Add Employee button

REQ-11: A button for each listed employee’s details (Details)

REQ-12: A button for each listed employee’s deletion (Delete)

REQ-13: A button for each listed employee’s information editing (Edit)

## Edit employee

4.4.1 Description and Priority

Edit a specific employee’s information. A feature only available to administrators.

Medium priority.

4.4.2 Stimulus/Response Sequences

Input: Employee’s full name, telephone, PID, address, employment, salary, a button that redirects to the employees’ list view, a submit button.

Output: Edit employee confirmation.

4.4.3 Functional Requirements

REQ-14: Full Name input-output textbox

REQ-15: Telephone input-output textbox

REQ-16: PID input-output textbox

REQ-17: Address input-output textbox

REQ-18: Employment input-output textbox

REQ-19: Salary input-output textbox

REQ-20: Back button

REQ-21: Submit button

## Add employee

4.5.1 Description and Priority

Add a new employee to the database. A feature only available to administrators.

Medium priority.

4.5.2 Stimulus/Response Sequences

Input: Employee’s full name, telephone, PID, address, employment, salary, a button that redirects to the employees’ list view, a submit button.

Output: Add employee confirmation.

4.5.3 Functional Requirements

REQ-22: Full Name input textbox

REQ-23: Telephone input textbox

REQ-24: PID input textbox

REQ-25: Address input textbox

REQ-26: Employment input textbox

REQ-27: Salary input textbox

REQ-28: Back button

REQ-29: Submit button

## Owners list

4.6.1 Description and Priority

Medium priority. Provides a report of amounts to be cashed from every owner of the association and every property owner’s information. High priority.

4.6.2 Stimulus/Response Sequences

Different views for each type of user logged in.

* Administrator:

Input: A button that redirects to the main page, a button to add a new owner to the database, delete buttons for every owner account details, and a button for editing personal account information.

Output: A list with owners’ information – full name, telephone, email, total payment amount for owned apartments.

* Typical user:

Input: A button that redirects to the main page and a button for editing personal account information.

Output: A list with owners’ information – full name, telephone, email, total payment amount for owned apartments.

4.6.3 Functional Requirements

REQ-30: Back to Main Page button

REQ-31: Add Owner button

REQ-32: A button for each listed owner’s deletion (Delete)

REQ-33: A button for editing personal account information (Edit)

## Edit owner

4.7.1 Description and Priority

Edit account information. A feature available to both administrators and typical users.

Low priority.

4.7.2 Stimulus/Response Sequences

Input: Owner’s full name, username, password, telephone, email, a button that redirects to the owners’ list view, a submit button.

Output: Edit owner account confirmation.

4.7.3 Functional Requirements

REQ-34: Full Name input-output textbox

REQ-35: Username input-output textbox

REQ-36: Password input-output textbox

REQ-37: Telephone input-output textbox

REQ-38: Email input-output textbox

REQ-39: Back button

REQ-40: Submit button

## Add owner

4.8.1 Description and Priority

Add a new owner account to the database. A feature only available to administrators.

Medium priority.

4.8.2 Stimulus/Response Sequences

Input: Owner’s full name, username, password, telephone, email, a button that redirects to the owners’ list view, a submit button.

Output: Add owner account confirmation.

4.8.3 Functional Requirements

REQ-41: Full Name input textbox

REQ-42: Username input textbox

REQ-43: Password input textbox

REQ-44: Telephone input textbox

REQ-45: Email input textbox

REQ-46: Back button

REQ-47: Submit button

## Apartment details

4.9.1 Description and Priority

View a specific apartment’s information. A feature available to both administrators and typical users.

Low priority.

4.9.2 Stimulus/Response Sequences

Input: A button that redirects to the main page.

Output: A list of selected apartment’s detailed information – owner’s name, apartment address, no. of tenants in the apartment, payment amount for the current month, debt from previous months, total payment amount.

4.9.3 Functional Requirements

REQ-48: Back to Main Page button

## Apartments owned list

4.10.1 Description and Priority

Provides a list of all apartments owned by a logged user and CRUD operations on their information. A feature available to both administrators and typical users.

High priority.

4.10.2 Stimulus/Response Sequences

Input: A button that redirects to the main page, a button to add a new apartment to the list, input-output boxes for editing apartments’ data (building, street, flat number, number of tenants) and a save button for saving it, a delete button, buttons for adding water consumption and adding payment receipts for all owned apartments in the list.

Output: Input-output boxes for editing apartments’ data, edit apartment confirmation.

4.10.3 Functional Requirements

REQ-49: Back to Main Page button

REQ-50: Add Apartment button

REQ-51: Building input-output textbox for every apartment in the list

REQ-52: Street input-output textbox for every apartment in the list

REQ-53: Flat No. input-output textbox for every apartment in the list

REQ-54: Tenants input-output textbox for every apartment in the list

REQ-55: Save button for every apartment in the list

REQ-56: Delete button for every apartment in the list

REQ-57: Add Water Consumption button for every apartment in the list

REQ-58: Add Payment Receipt button for every apartment in the list

## Add payment receipt

4.11.1 Description and Priority

Add a payment receipt for a specific apartment owned by a logged in user. A feature available to both administrators and typical users.

High priority.

4.11.2 Stimulus/Response Sequences

Input: A button that redirects to the owned apartments’ list, date of the receipt, and amount payed, a submit button.

Output: Add new receipt confirmation.

4.11.3 Functional Requirements

REQ-59: Back button

REQ-60: Date input textbox

REQ-61: Amount input textbox

REQ-62: Submit button

## Add water consumption

4.12.1 Description and Priority

Add water consumption for a specific apartment owned by a logged in user. A feature available to both administrators and typical users.

High priority.

4.12.2 Stimulus/Response Sequences

Input: A button that redirects to the owned apartments’ list, date, cold water and warm water consumed (units), a submit button.

Output: Add water consumption confirmation.

4.12.3 Functional Requirements

REQ-63: Back button

REQ-64: Warm water input textbox (units)

REQ-65: Cold water input textbox (units)

REQ-66: Date input textbox

REQ-67: Submit button

## Add apartment

4.13.1 Description and Priority

Add a new apartment to the list of apartments owned by a logged in user. A feature available to both administrators and typical users.

High priority.

4.13.2 Stimulus/Response Sequences

Input: A button that redirects to the owned apartments’ list, building, street, flat number, number of tenants, a submit button.

Output: Add new apartment confirmation.

4.13.3 Functional Requirements

REQ-68: Back button

REQ-69: Building input textbox

REQ-70: Street input textbox

REQ-71: Flat No. input textbox

REQ-72: Tenants input textbox

REQ-73: Submit button

## Contracts list

4.14.1 Description and Priority

Provides a list with all available contracts and possibility to add new contracts or finalize existing ones. A feature only available to administrators.

High priority.

4.14.2 Stimulus/Response Sequences

Input: A button which redirects to the main page, a button for adding a new contract, details and finalize buttons for every listed contract.

Output: A list with contracts’ information – date, type and provider.

4.14.3 Functional Requirements

REQ-74: Back to Main Page button

REQ-75: Add Contract button

REQ-76: Details button for every contract in the list

REQ-77: Finalize button for every contract in the list

## Add contract

4.15.1 Description and Priority

Add a new contract to the existing contracts list. A feature only available to administrators.

High priority.

4.15.2 Stimulus/Response Sequences

Input: A button which redirects to the contracts’ list view, date, type, provider, price per unit, and a submit button.

Output: Add new contract confirmation.

4.15.3 Functional Requirements

REQ-78: Back button

REQ-79: Date input box

REQ-80: Type dropdown (dropdown for contract category)

REQ-81: Provider input textbox

REQ-82: Price input textbox

REQ-83: Submit button

## Contract details

4.16.1 Description and Priority

View details for a specified contract from the contracts’ list.

A feature only available to administrators.

Low priority.

4.16.2 Stimulus/Response Sequences

Input: A button which redirects to the contracts’ list view and a finalize button.

Output: Contract information – date, type (category), provider and price per unit.

4.16.3 Functional Requirements

REQ-84: Back button

REQ-85: Finalize button

# Other Nonfunctional Requirements

## Performance Requirements

The average server response time should be between 0.5 and 1 second.

The server response time upper limit should be 5 seconds.

## Safety Requirements

Not applicable.

## Security Requirements

The password will be crypted with a hash key.

The password set by the user must have at least one uppercase character, one lowercase character, non-letter or digit character. The minimum length of the password must be at least 6 characters long.

## Software Quality Attributes

The 10 MISRA coding rules used in development are:

1. Unused method parameters should be removed **(MISRA C:2012, 2.7)**
2. Destructors should not throw exceptions **(MISRA C++:2008, 15-5-1)**
3. Multiple variables should not be declared on the same line **(MISRA C++:2008, 8-0-1)**
4. Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression **(MISRA C:2012, 13.3)**
5. Useless "if(true) {...}" and "if(false){...}" blocks should be removed **(MISRA C:2012, 14.3)**
6. Control structures should use curly braces **(MISRA C:2012, 15.6)**
7. Sections of code should not be commented out **(MISRA C:2012, Dir. 4.4)**
8. Literal suffixes should be upper case **(MISRA C:2012, 7.3)**
9. "goto" statement should not be used **(MISRA C:2012, 15.1)**
10. Boolean expressions should not be gratuitous **(MISRA C:2012, 14.3)**

## Business Rules

Not applicable.

# Other Requirements

Appendix A: Glossary

A list of all acronyms and abbreviations used in the SRS document:

1. **POAM** – **P**roperty **O**wners **A**ssociation **M**anager (the name given to the application).
2. **CRUD** – an acronym for **C**reate, **R**ead, **U**pdate and **D**elete.
3. **MISRA** - **M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation.
4. **r/w** – an acronym for read-write