

Dipartimento di Ingegneria Elettrica e Tecnologie dell’Informazione  
Corso di Laurea in Informatica

**Software Engineering I Project**

GCI ‘16

**Customer:**

Sergio Di Martino

**Team:**

GCI16\_25

Carlo De Vita N86001634

Mattia Iodice N86001672

Riccardo Grieco N86001614

Table of contents

[1 Introduction 5](#_Toc495863483)

[1.1 Subject 5](#_Toc495863484)

[1.2 Revision history 7](#_Toc495863485)

[2 Team Organization 8](#_Toc495863486)

[2.1 Activity planning 8](#_Toc495863487)

[2.1.1 Gantt Diagram 8](#_Toc495863488)

[2.1.2 Pert Diagram 8](#_Toc495863489)

[2.2 Resources sharing 8](#_Toc495863490)

[3 Requirements document 9](#_Toc495863491)

[3.1 Requirements List 9](#_Toc495863492)

[3.1.1 Functional requirements 9](#_Toc495863493)

[3.1.2 Non-functional Requirements 9](#_Toc495863494)

[3.1.3 Domain Requirements 9](#_Toc495863495)

[3.2 Use Case Diagrams 10](#_Toc495863496)

[3.2.1 Back office operator use case 10](#_Toc495863497)

[3.2.2 Readings operator use case 10](#_Toc495863498)

[3.2.3 Administrator use case 11](#_Toc495863499)

[3.3 Cockburn Diagrams for Use Cases 12](#_Toc495863500)

[3.3.1 Create payment order 12](#_Toc495863501)

[3.3.2 Delete payment order 14](#_Toc495863502)

[3.3.3 Issue payment order 16](#_Toc495863503)

[3.3.4 Save as not pertinent payment order 18](#_Toc495863504)

[3.3.5 Reissue payment order 20](#_Toc495863505)

[3.3.6 Saves payment order as paid 22](#_Toc495863506)

[3.3.7 Saves a payment order as suspended 24](#_Toc495863507)

[3.3.8 Sends meter readings 26](#_Toc495863508)

[3.4 User Interface Mockups 28](#_Toc495863509)

[3.4.1 Login 28](#_Toc495863510)

[3.4.2 Main Menu 29](#_Toc495863511)

[3.4.3 Payment Order 30](#_Toc495863512)

[3.4.4 Candidate Payment Orders 30](#_Toc495863513)

[3.4.5 Confirm 31](#_Toc495863514)

[3.4.6 Operation Success 31](#_Toc495863515)

[3.4.7 Readings Main 32](#_Toc495863516)

[3.4.8 Save Reading 33](#_Toc495863517)

[3.4.9 Confirm Reading 34](#_Toc495863518)

[3.4.10 Successful Sending 35](#_Toc495863519)

[3.4.11 Failed Sending 36](#_Toc495863520)

[3.5 Entity-Boundary-Control Class Diagram 37](#_Toc495863521)

[3.5.1 Desktop application controllers 38](#_Toc495863522)

[3.5.2 Mobile application controller 40](#_Toc495863523)

[3.6 Sequence Diagrams 41](#_Toc495863524)

[3.6.1 Create payment order 41](#_Toc495863525)

[3.6.2 Delete payment order 42](#_Toc495863526)

[3.6.3 Issue payment order 43](#_Toc495863527)

[3.6.4 Save payment order as not pertinent 44](#_Toc495863528)

[3.6.5 Reissue payment order 45](#_Toc495863529)

[3.6.6 Save payment order as paid 46](#_Toc495863530)

[3.6.7 Save payment order as suspended 47](#_Toc495863531)

[3.6.8 Send readings 48](#_Toc495863532)

[3.7 Payment Order Statechart Diagram 49](#_Toc495863533)

[4 Design documentation 50](#_Toc495863534)

[4.1 System design 50](#_Toc495863535)

[4.1.1 Software architecture 50](#_Toc495863536)

[4.1.2 Client server communication 51](#_Toc495863537)

[4.2 Technologies 51](#_Toc495863538)

[4.2.1 Frameworks and Libraries 52](#_Toc495863539)

[4.3 Object design 53](#_Toc495863540)

[4.3.1 Class diagrams 53](#_Toc495863541)

[4.3.2 Sequence diagrams 57](#_Toc495863542)

[4.4 CRC cards 65](#_Toc495863543)

[4.4.1 Server 65](#_Toc495863544)

[4.4.2 Back-Office Application 66](#_Toc495863545)

[4.4.3 Mobile Application 69](#_Toc495863546)

[5 Testing documentation 71](#_Toc495863547)

[5.1 System Testing 71](#_Toc495863548)

[5.1.1 Back-Office Application 71](#_Toc495863549)

[5.1.2 Mobile Application 73](#_Toc495863550)

[5.2 Unit Testing 75](#_Toc495863551)

[5.2.1 Parameter equivalence classes 75](#_Toc495863552)

[5.2.2 Test cases 75](#_Toc495863553)

[5.2.3 Code 76](#_Toc495863554)

[5.2.4 jUnit test output 81](#_Toc495863555)

[6 Glossary (A-Z) 82](#_Toc495863556)

# Introduction

## Subject

This document represents the documentation with requirements collection, design and testing of the software product GCI ‘16, the analysis is based on a functionality subset selected by the costumer, Sergio Di Martino. In particular we have developed these points: 3, 4 and 6. For more details we have reported the specifications of the product as taken from “Progetto-16-17\_V03F” document, published by the customer below.

|  |  |
| --- | --- |
| SoftEngUniNA | Protocollo N.: |
| **3.** | **SPECIFICHE DI PRODOTTO/ SERVIZIO** |

Il ***GCI16*** (Gestione Canoni Idrici) è un Sistema Informativo di una Pubblica Amministrazione Locale, finalizzato a gestire il pagamento di una tassa per il consumo di acqua. Il sistema distribuito presenta una parte di Back-Office per la gestione di contribuenti, ingiunzioni e pagamenti, ed un client su dispositivo mobile, utilizzato dai Controllori per leggere i consumi di acqua presso i contatori.

I principali servizi offerti dal sistema sono i seguenti:

1. Gestione Anagrafica Contribuenti.
2. Generazione Bollette.
3. Gestione Ingiunzioni di Pagamento.
4. Emissione Ingiunzioni di Pagamento.
5. Amministrazione Sistema
6. Lettura Consumi

Gli operatori del sistema, opportunamente autenticati, devono avere la possibilità di effettuare le tipiche operazioni CRUD sulla base di dati contenente l’anagrafica dei contribuenti. Gli operatori possono, ogni trimestre, chiedere la generazione delle bollette per tutte le utenze. Gli operatori possono inoltre creare nuove ingiunzioni di pagamento, specificando una persona fisica o giuridica che risulta essere evasore, un anno di competenza e inserendo le letture prese dai tecnici per il contatore corrispondente. Il sistema, basandosi su alcuni parametri configurabili, deriva l'ammontare dovuto. Un'ingiunzione può essere modificata ed eliminata solo se non emessa, previa opportuna ricerca.

Per emettere un’ingiunzione di pagamento, un operatore del Back-Office può selezionarne una da una lista di ingiunzioni non ancora emesse e, previo inserimento di un numero di protocollo, procedere con l'emissione. Ciò deve generare un documento in formato PDF contenente le comunicazioni che saranno inviate ai contribuenti.

Le ingiunzioni emesse non possono più essere cancellate, ma solo aggiornate. In presenza del relativo pagamento, l’ingiunzione viene archiviata. In presenza di un’opposizione, l’ingiunzione viene sospesa.

Inoltre è possibile effettuare il re-inserimento e quindi la riemissione di quelle ingiunzioni che non è stato possibile notificare (ad esempio per indirizzo errato).

Un amministratore di sistema deve inoltre poter gestire gli operatori, specificare l'ammontare delle spese di spedizione, delle tariffe per metro cubo e la percentuale di IVA da applicare. Infine, gli addetti alla lettura dei consumi utilizzano un dispositivo mobile per specificare, ogni 3 mesi, i consumi d’acqua di un utente, così come rilevabile dai contatori. Il dispositivo invia la lettura al server in modalità wireless.

Le funzionalità da 1 a 4 sono disponibili sul Back-Office ad un operatore opportunamente identificato dal sistema.

La funzionalità 5 è disponibile in forma di App sul client mobile dato in dotazione agli Operatori sul Campo.

**Dato questo insieme di funzionalità, il Committente deciderà insieme al fornitore un sottoinsieme di caratteristiche da modellare e implementare, previo incontro formale secondo le modalità specificate sul sito web.**

SoftEngUniNA Protocollo N.:

Per lo svolgimento della suddetta attività è obbligatorio l’utilizzo di un tool di CASE.

Si richiede tassativamente di astrarre il design per favorire il riutilizzo del codice e la futura implementazione di altre funzionalità.

**Nota:** Tutte le attività devono essere effettuate presso il fornitore tranne per i momenti di condivisione definiti.

## Revision history

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| 07/12/2016 | 0.1 | Created first requirements list (functional, non-functional and domain) and Use Cases. |
| 31/03/2017 | 0.1.1 | Updated the requirements list and created Mockups. |
| 13/04/2017 | 0.2 | Updated Use Cases and Mockups. |
| 29/04/2017 | 0.3 | Created Cockburn diagrams for use cases. |
| 3/05/2017 | 0.3.1 | Added the Glossary and the State chart diagram for payment orders. |
| 10/05/2017 | 0.3.2 | Updated and finished Use case, Mockups and Cockburn diagrams. |
| 25/05/2017 | 0.4 | Added Entity Boundary Control Class Diagram. |
| 11/06/2017 | 0.4.1 | Updated Entity Boundary Control and added Sequence Diagrams. |
| 14/06/2017 | 0.5 | Created System design section. |
| 22/07/2017 | 0.5.1 | Created Object design section with the Object Design Class Diagrams. |
| 26/07/2017 | 0.5.2 | Fixed Object Design Class Diagrams. |
| 27/07/2017 | 0.5.3 | Added technologies and libraries for some functionalities. |
| 30/07/2017 | 0.5.4 | Added CRC Cards and Sequence Diagrams. |
| 7/09/2017 | 0.6 | Updated Sequence Diagrams completing Design section and created Testing section. |
| 22/09/2017 | 0.6.1 | Added System testing plan. |
| 8/10/2017 | 0.6.2 | Added Unit Testing and completed the first version of the documentation. |
| 14/10/2017 | 1.0 | Revision of the entire documentation and completed the first version. |

# Team Organization

## Activity planning

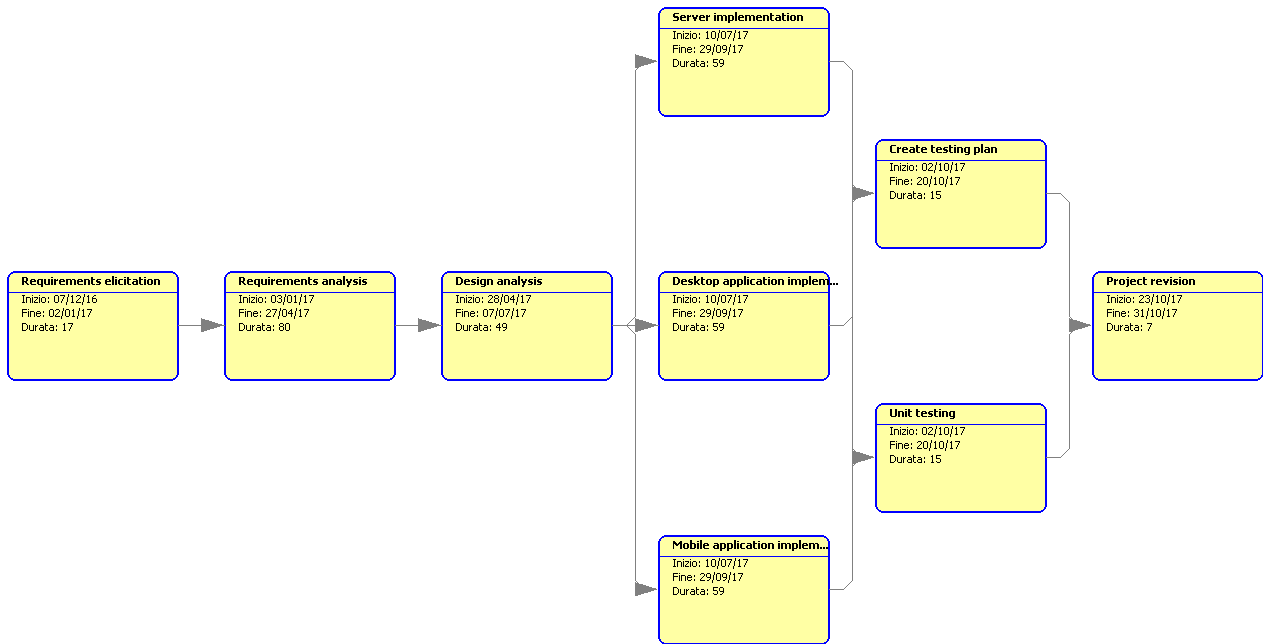
The following diagrams show how the team has decided to organize the teamwork. They have been developed in the early stage of the project and deadlines first decided could not have been really respected.

The diagrams have been created using Gantt Project Tool.

### Gantt Diagram



### Pert Diagram



## Resources sharing

The resources are shared using [Git](https://git-scm.com/) version control system, under the hosting platform [Github](https://github.com/).

The link of the resources is <https://github.com/CarloDeVita/INGSW.git>

# Requirements document

## Requirements List

### Functional requirements

* The Back-Office Operator must be able to create new payment orders, specifying a bill from a list of candidates.
* The Back-Office Operator can issue not-issued payment orders.
* When a payment order is issued, it is given a protocol number generated incrementally.
* The Back-Office Operator can delete not-issued payment orders.
* The Back-Office Operator can reissue suspended payment orders.
* When a payment order is reissued his protocol number is not changed.
* The Back-Office Operator can save as not pertinent suspended payment orders.
* The Back-Office Operator can save as paid notified payment orders.
* The Back-Office Operator can save as suspended notified payment orders.
* When a payment order is issued the system must generate and store a PDF document with all the useful communications for the debtor.
* A user must be able to log in as a Back-Office Operator in desktop application.
* A user must be able to log in as a Readings Operator in mobile application.
* The Readings Operator can download the list of his assignments from the mobile application.
* The Readings Operator can save readings specifying a meter and the water consumed.
* The Readings Operator must be able to send saved readings via Internet.

### Non-functional Requirements

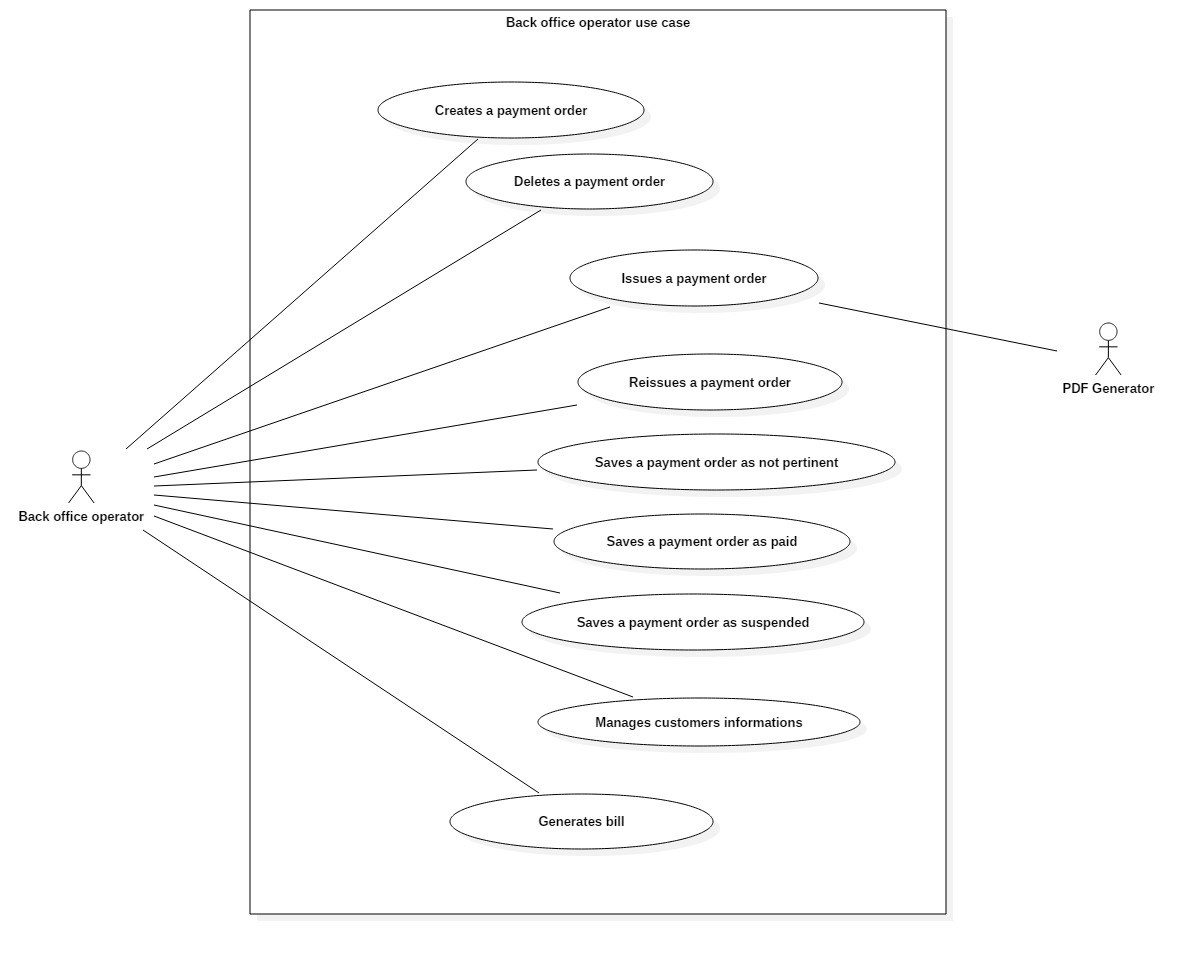
* Payment orders can be seen and searched specifying a protocol, a debtor, a year, a trimester and/or a status.
* The readings are first saved locally on smartphone and sent later only if internet connection is available.
* To use the system every user must log in.

### Domain Requirements

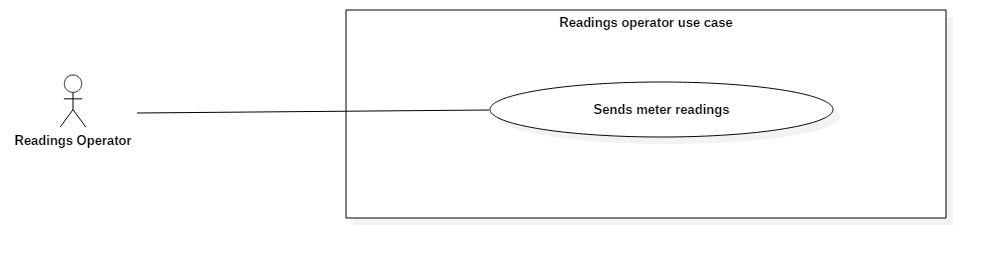
* Not issued payment orders can only be issued or deleted.
* Issued payment orders can only be archived.
* Once saved as paid or saved as not pertinent, a payment order must be archived.
* Suspended payment orders can only be reissued or saved as not pertinent.
* Every issued payment order is identified by a protocol number.
* Notified payment orders can only be saved as paid or as suspended.
* The considered unit of measurement of water consumption is m3.

## Use Case Diagrams

### Back office operator use case



### Readings operator use case



### Administrator use case



## Cockburn Diagrams for Use Cases

### Create payment order

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#1* | *Create payment order* | | |
| Goal in Context | Generate a new payment order. | | |
| Scope & Level | System under design; Level = User goal | | |
| Preconditions | The user must be logged in the system. | | |
| Success End Condition | A payment order is created and stored in the system. | | |
| Failed End Condition | The user presses “Home” in “Payment Orders” mockup. | | |
| Primary Actor | Back Office Operator. | | |
| Trigger | The operator presses “Payment Orders” in “Main Menu” mockup. | | |
| DESCRIPTION | **Step n°** | **Back Office Operator** | **System** |
| *1* |  | Shows mockup “Payment Orders” |
| *2* | Presses “New payment order” |  |
| *3* |  | Shows mockup “Candidate Payment Orders” |
| *4* | Selects a bill |  |
| *5* |  | Enables “Create Payment Order” |
| *6* | Presses “Create Payment Order” |  |
| *7* |  | Shows mockup “Confirm” |
| *8* | Presses “Yes” |  |
| *9* |  | Shows mockup “Operation Success” |
| *10* | Presses “OK” |  |
|  | *11* |  | *Returns to mockup “Payment Orders”* |

|  |  |  |  |
| --- | --- | --- | --- |
| EXTENSIONS | **Step n°** | **Back Office Operator** | **System** |
|  | *2.1* | Presses “Home” |  |
|  | *2.2* |  | *Shows mockup “Main Menu”* |
| SUBVARIATIONS | *Step* | **Back Office Operator** | **System** |
|  | *2.2* | Sets filtering parameters and presses “Filter” |  |
|  | *3.2* |  | *Shows the filtered table and return to step #2* |

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#2* | *Delete payment order* | | |
| Goal in Context | Delete a selected payment order. | | |
| Scope & Level | System under design; Level = User goal | | |
| Preconditions | The user must be logged in the system. | | |
| Success End Condition | A payment order is deleted from the system. | | |
| Failed End Condition | The user presses “Home” in “Payment Orders” mockup. | | |
| Primary Actor | Back Office Operator. | | |
| Trigger | The operator presses “Payment Orders” in “Main Menu” mockup. | | |
| DESCRIPTION | **Step n°** | **Back Office Operator** | **System** |
| *1* |  | Shows mockup “Payment Orders” |
| *2* | Selects a payment order |  |
| *3* |  | Enables “Delete” |
| *4* | Presses “Delete” |  |
| *5* |  | Shows mockup “Confirm” |
| *6* | Presses “Yes” |  |

### Delete payment order

|  |  |  |  |
| --- | --- | --- | --- |
|  | *7* |  | Shows mockup “Operation Success” |
| *10* | Presses “OK” |  |
|  | *11* |  | *Returns to mockup “Payment Orders”* |
| EXTENSIONS | **Step n°** | **Back Office Operator** | **System** |
|  | *2.1* | Presses “Home” |  |
|  | *3.1* |  | *Shows mockup “Main Menu”* |
| SUBVARIATIONS | *Step* | **Back Office Operator** | **System** |
|  | *2.2* | Sets filtering parameters and presses “Filter” |  |
|  | *3.2* |  | *Shows the filtered table and return to step #2* |

### Issue payment order

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#3* | Issue payment orders. | | |
| Goal in Context | User issues a payment order. | | |
| Scope & Level | System Under Design; Level = “User goal” | | |
| Preconditions | User must be logged. | | |
| Success End Condition | A payment order is issued. | | |
| Failed End Condition | User presses ‘Home’ in mockup ‘Payment Orders’. | | |
| Primary Actor | Back office operator. | | |
| Trigger | Back office operator presses ‘Payment Orders’ in mockup ‘Main menu’. | | |
| DESCRIPTION | **Step n°** | **Back Office Operator** | **System** |
| *1* | Presses ‘Payment Orders’ in mockup ‘Main’. |  |
| *2* |  | Shows mockup ‘Payment Orders’. |
| *3* | Selects a not issued payment order. |  |
| *4* |  | Enables button ‘Issue’. |
| *5* | Presses ‘Issue’. |  |
| *6* |  | Shows mockup ‘Confirm’. |
| *7* | Presses ‘Yes’. |  |
| *8* |  | Shows mockup ‘Successful operation’. |
| *9* | Presses ‘Ok’. |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | *10* |  | Shows mockup ‘Payment Orders’ in which the payment order concerned shall be issued, and UC successfully ends. |
| EXTENSIONS | ***Step*** | **Back Office Operator** | **System** |
|  | *\*.1* | Presses ‘Home’ in mockup ‘Payment Orders’. |  |
| *\*2.1* |  | Shows mockup ‘Main’ and UC fails. |
|  | *5.2* | Presses ‘No. |  |
| *6.2* |  | Back to step 2. |
| SUBVARIATIONS | ***Step*** | **Back Office Operator** | **System** |
|  | *3.3* | Filters a not issued payment order, selects. |  |
| *4.3* |  | Back to step 4. |

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#4* | Save as not pertinent payment orders. | | |
| Goal in Context | User saves as not pertinent a suspended payment order. | | |
| Scope & Level | System Under Design; Level = “User goal” | | |
| Preconditions | User must be logged. | | |
| Success End Condition | A payment order is saved as not pertinent. | | |
| Failed End Condition | User presses ‘Home’ in mockup ‘Payment Orders’. | | |
| Primary Actor | Back office operator. | | |
| Trigger | Back office operator presses ‘Payment Orders’ in mockup ‘Main menu’. | | |
| DESCRIPTION | **Step n°** | **Back Office Operator** | **System** |
| *1* | Presses ‘Payment Orders’ in mockup ‘Main menu’. |  |
| *2* |  | Shows mockup ‘Payment Orders’. |
| *3* | Selects a suspended payment order. |  |
| *4* |  | Enables button ‘Save as not pertinent’. |
| *5* | Presses ‘Save as not pertinent’. |  |
| *6* |  | Shows mockup ‘Confirm’. |
| *7* | Presses ‘Yes’. |  |
| *8* |  | Shows mockup ‘Successful operation’. |
| *9* | Presses ‘Ok’. |  |

### Save as not pertinent payment order

|  |  |  |  |
| --- | --- | --- | --- |
|  | *10* |  | Shows mockup ‘Payment Orders’ in which there isn’t the payment order concerned because it shall be saved as not pertinent, and UC successfully ends. |
| EXTENSIONS | ***Step*** | **Back Office Operator** | **System** |
|  | *\*.1* | Presses ‘Home’ in mockup ‘Payment Orders’. |  |
| *\*2.1* |  | Shows mockup ‘Main menu’ and UC fails. |
|  | *5.2* | Presses ‘No. |  |
| *6.2* |  | Back to step 2. |
| SUBVARIATIONS | ***Step*** | **Back Office Operator** | **System** |
|  | *3.3* | Filters suspended payment order, selects it. |  |
| *4.3* |  | Back to step 4. |

### Reissue payment order

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#5* | Reissue payment orders. | | |
| Goal in Context | User reissues a suspended payment order. | | |
| Scope & Level | System Under Design; Level = “User goal” | | |
| Preconditions | User must be logged. | | |
| Success End Condition | A payment order is reissued. | | |
| Failed End Condition | User presses ‘Home’ in mockup ‘Payment Orders’. | | |
| Primary Actor | Back office operator. | | |
| Trigger | Back office operator presses ‘Payment Orders’ in mockup ‘Main menu’. | | |
| DESCRIPTION | **Step n°** | **Back Office Operator** | **System** |
| *1* | Presses ‘Payment Orders’ in mockup ‘Main menu’. |  |
| *2* |  | Shows mockup ‘Payment Orders’. |
| *3* | Selects a suspended payment order. |  |
| *4* |  | Enables button ‘Reissue’. |
| *5* | Presses ‘Reissue’. |  |
| *6* |  | Shows mockup ‘Confirm’. |
| *7* | Presses ‘Yes’. |  |
| *8* |  | Shows mockup ‘Successful operation’. |
| *9* | Presses ‘Ok’. |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | *10* |  | Shows mockup ‘Payment Orders’ in which the payment order concerned shall be issued, and UC successfully ends. |
| EXTENSIONS | ***Step*** | **Back Office Operator** | **System** |
|  | *\*.1* | Presses ‘Home’ in mockup ‘Payment Orders’. |  |
| *\*2.1* |  | Shows mockup ‘Main menu’ and UC fails. |
|  | *5.2* | Presses ‘No. |  |
| *6.2* |  | Back to step 2. |
| SUBVARIATIONS | ***Step*** | **Back Office Operator** | **System** |
|  | *3.3* | Filters suspended payment order, selects it. |  |
| *4.3* |  | Back to step 4. |

### Saves payment order as paid

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#6* | *Saves payment order as paid* | | |
| Goal in Context | Saving a payment order as paid | | |
| Scope & Level | System under design; Level = User goal | | |
| Preconditions | User must be logged in. | | |
| Success End Condition | The payment order is saved in the system as paid. | | |
| Failed End Condition | There is no payment order that can be saved as paid. The user interrupts the use case. | | |
| Primary Actor | Back Office Operator. | | |
| Trigger | The operator presses button “Payment Order” in “Main Menu” mockup. | | |
| DESCRIPTION | **Step n°** | **Back Office Operator** | **System** |
|  | *1* |  | Shows mockup “Payment Orders” |
| *2* | Selects a “notified” payment order from the table and presses “Save as paid” button |  |
| *3* |  | Shows mockup “Confirm” |
|  | *4* | Presses OK |  |
| *5* |  | Shows mockup “Operation Success” |
| *6* | Presses OK |  |
| *7* |  | *Returns to mockup “Payment Orders”* |

|  |  |  |  |
| --- | --- | --- | --- |
| EXTENSIONS | **Step n°** | **Back Office Operator** | **System** |
|  | *2a* | Presses “Home” |  |
|  | *3a* |  | *Shows mockup “Main Menu”* |
| SUBVARIATIONS | ***Step n°*** | **Back Office Operator** | **System** |
|  | *2b* | Sets filtering parameters and presses “Filter” |  |
| *3b* |  | *Shows the filtered table and return to step #2* |

### Saves a payment order as suspended

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#7* | *Saves a payment order as suspended.* | | |
| Goal in Context | Saving a payment order as suspended. | | |
| Scope & Level | System under design; Level = User goal. | | |
| Preconditions | User must be logged in. | | |
| Success End Condition | The payment order is saved in the system as suspended. | | |
| Failed End Condition | There is no payment order that can be saved as suspended. The user interrupts the use case. | | |
| Primary Actor | Back Office Operator. | | |
| Trigger | The operator presses button “Payment Order” in “Main Menu”. | | |
| DESCRIPTION | **Step n°** | **Back Office Operator** | **System** |
|  | *1* |  | Shows mockup “Payment Orders” |
| *2* | Selects a “notified” payment order from the table and presses “Save as suspended” button |  |
| *3* |  | Shows mockup “Confirm” |
| *4* | Presses OK |  |
| *5* |  | Shows mockup “Operation Success” |
| *6* | Presses OK |  |
| *7* |  | *Returns to mockup “Payment Order”* |

|  |  |  |  |
| --- | --- | --- | --- |
| EXTENSIONS | **Step n°** | **Back Office Operator** | **System** |
|  | *2a* | Presses “Home” |  |
|  | *3a* |  | *Shows mockup “Main Menu”* |
| SUBVARIATIONS | ***Step n°*** | **Back Office Operator** | **System** |
|  | *2b* | Sets filtering parameters and presses “Filter” button |  |
| *3b* |  | *Shows the filtered table and return to step #2* |

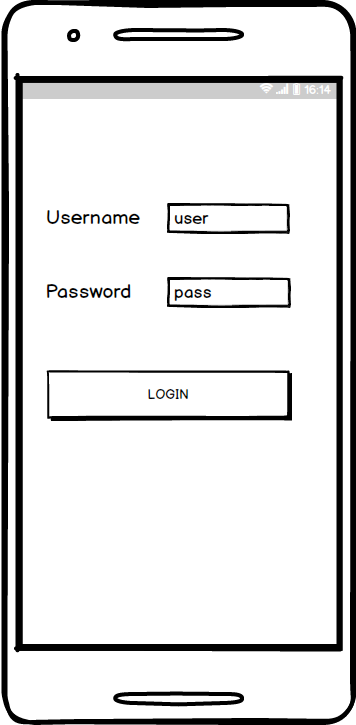
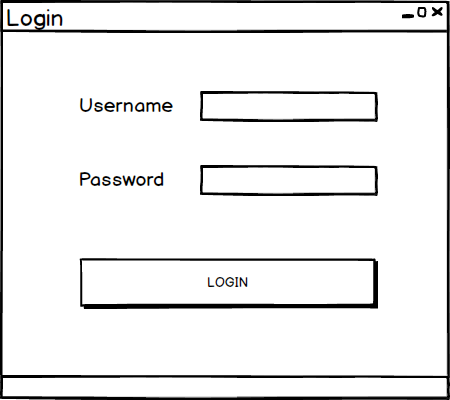
### Sends meter readings

|  |  |  |  |
| --- | --- | --- | --- |
| USE CASE *#8* | *Sends meter readings.* | | |
| Goal in Context | Reading the water consumption of a customer. | | |
| Scope & Level | System under design; Level = User goal | | |
| Preconditions | User must be logged in. | | |
| Success End Condition | The Reading Operator saves the water consumption and sends it to the server. | | |
| Failed End Condition | The Readings Operator closes the application.  The Readings Operator doesn’t have readings left to do.  There is no Internet connection to send the readings done. | | |
| Primary Actor | Readings Operator | | |
| Trigger | User logs in the mobile app. | | |
| DESCRIPTION | **Step n°** | **Readings Operator** | **System** |
|  | *1* |  | Shows mockup “Readings Main” |
| *2* | Selects a customer from the list and clicks “Save reading” button |  |
| *3* |  | Shows mockup “Save Reading” |
| *4* | Inserts the amount of water consumed and presses “Save” button |  |
| *5* |  | Shows mockup “Confirm Reading” containing all the reading’s information |
| *6* | Presses “Yes” button |  |
| *7* |  | *Returns to mockup “Readings Main”* |
| *8* | Presses “Send readings” button |  |

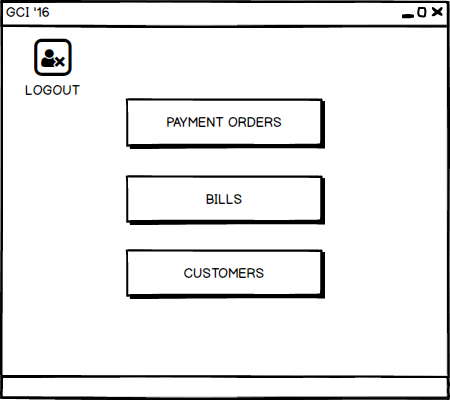
|  |  |  |  |
| --- | --- | --- | --- |
|  | *9* |  | *Shows mockup “Successful Sending”* |
| *10* | Presses OK |  |
| *11* |  | *Returns to mockup “Readings Main”* |
| EXTENSIONS | **Step n°** | **Readings Operator** | **System** |
|  | *9a* |  | *Shows mockup “Failed Sending”* |
| *10a* | Presses OK |  |
| *11a* |  | *Returns to mockup “Readings Main”* |
| SUBVARIATIONS | ***Step n°*** | **Readings Operator** | **System** |
|  | *8b* | Returns to step #1 |  |

## User Interface Mockups

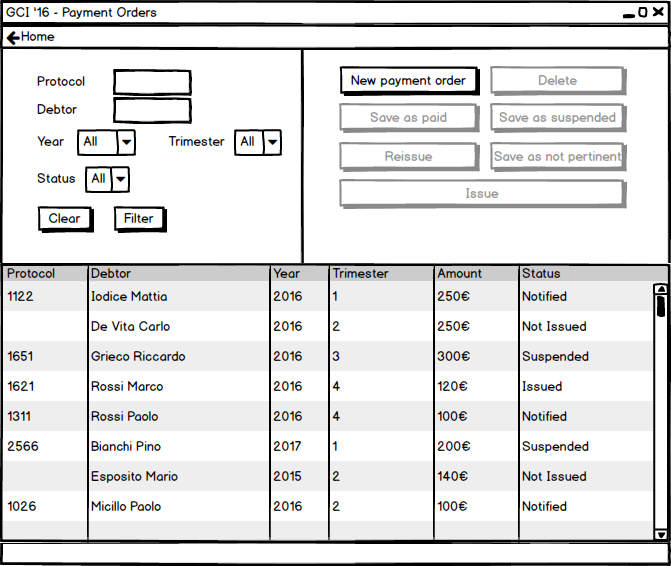
### Login



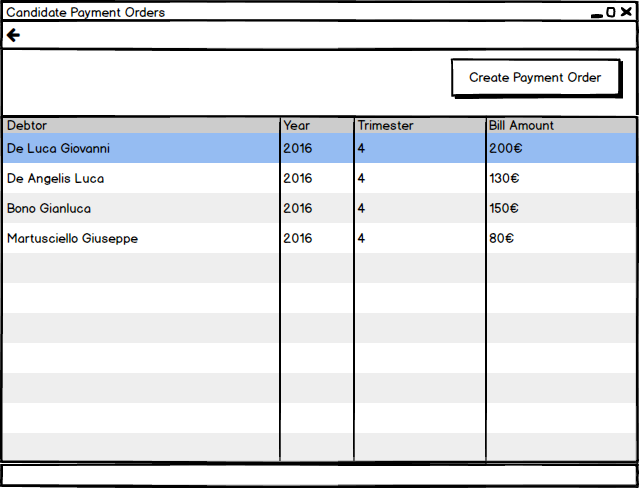
### Main Menu



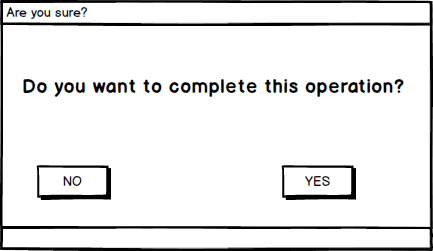
### Payment Order



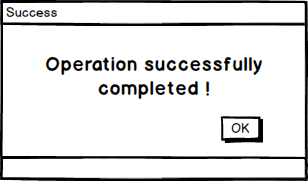
### Candidate Payment Orders



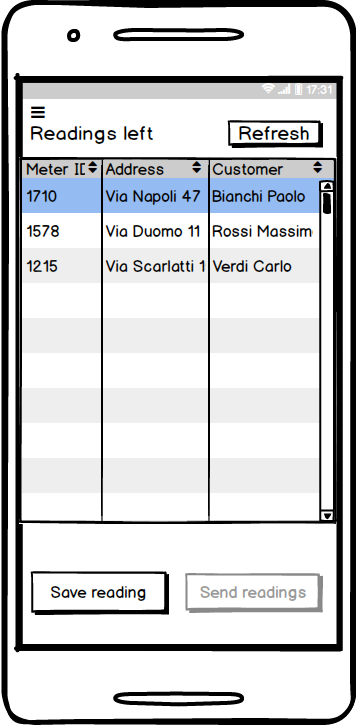
### Confirm



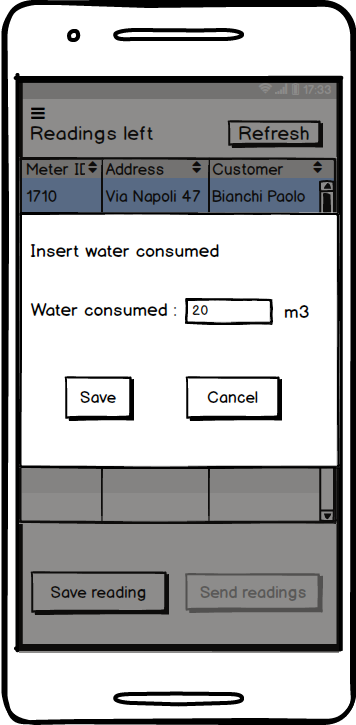
### Operation Success



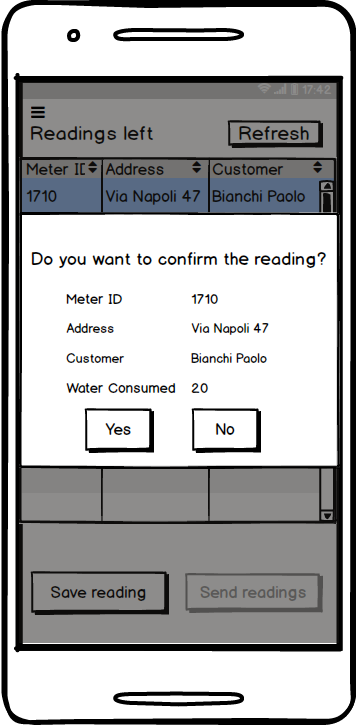
### Readings Main



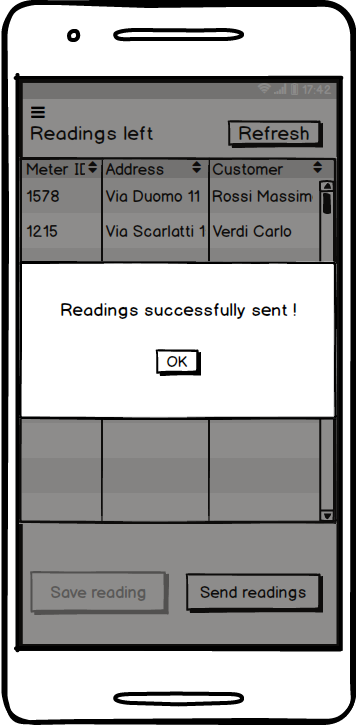
### Save Reading



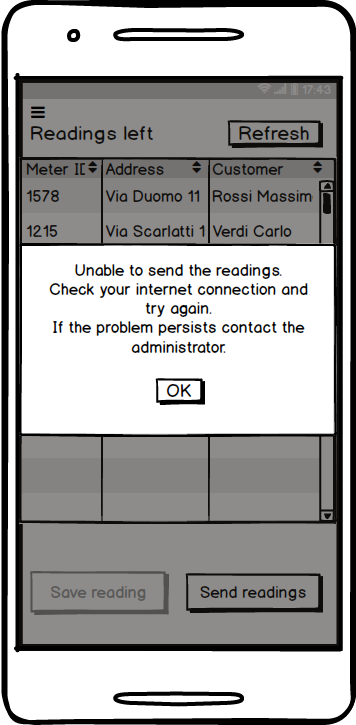
### Confirm Reading



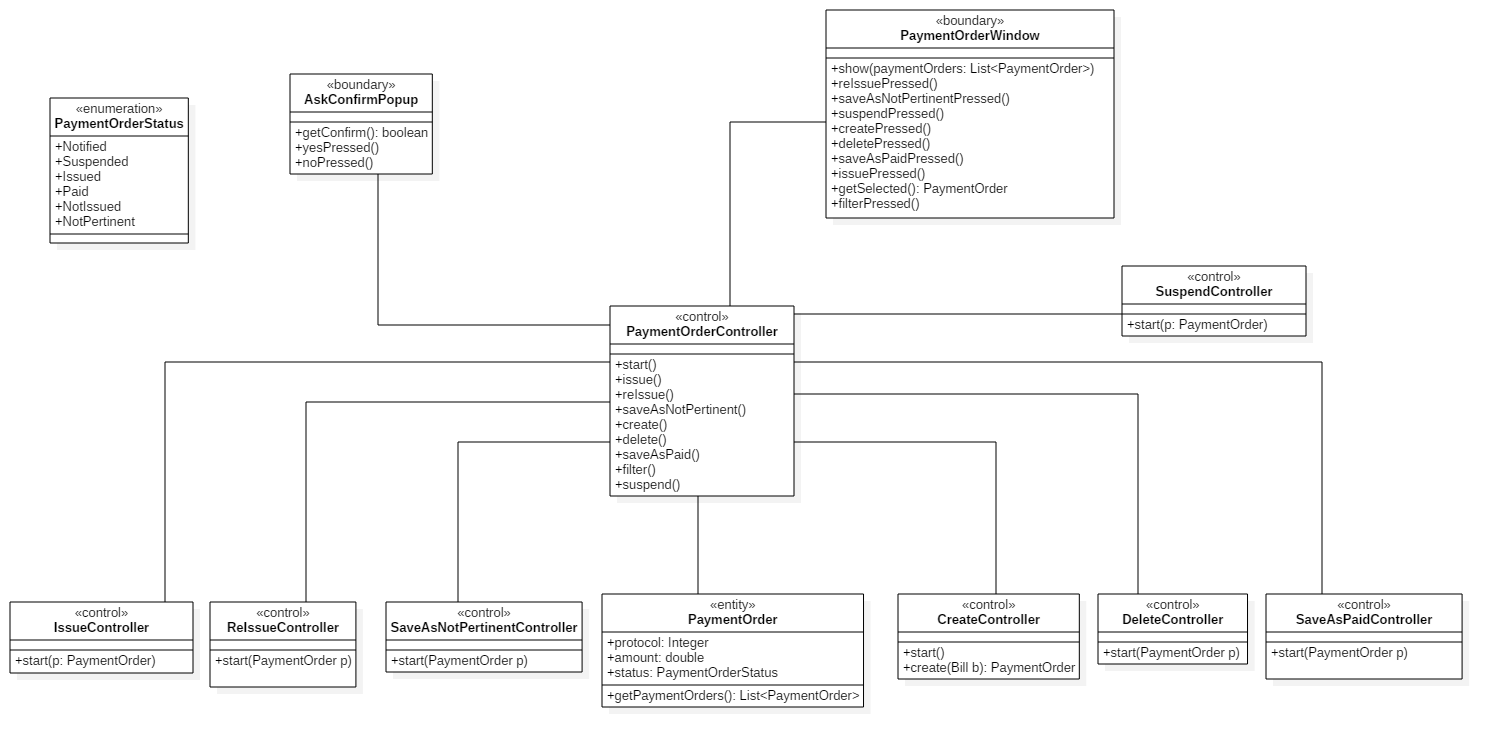
### Successful Sending



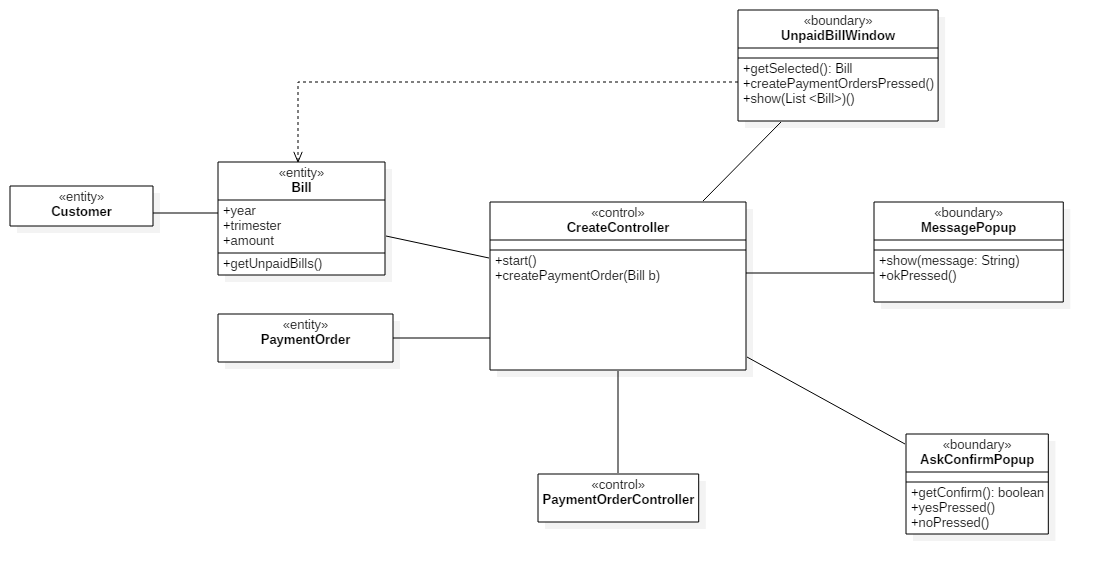
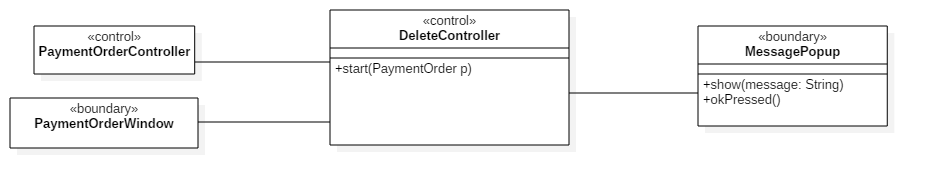
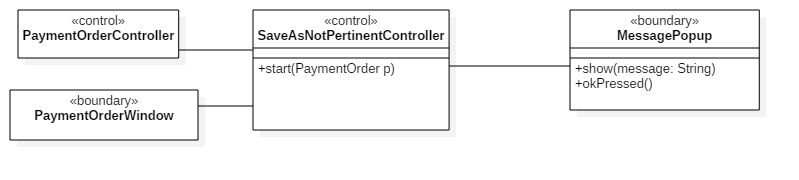
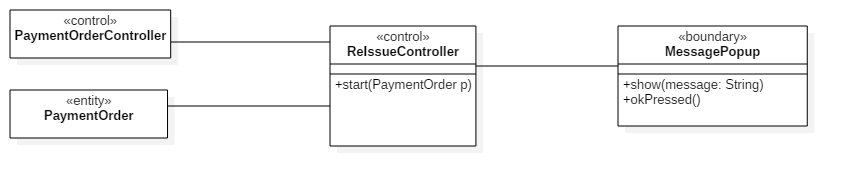
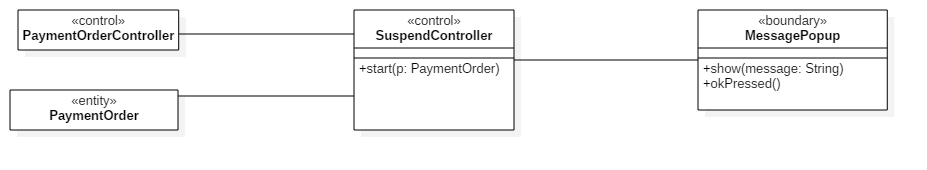
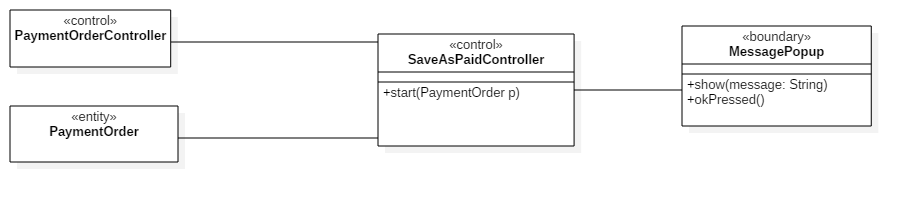
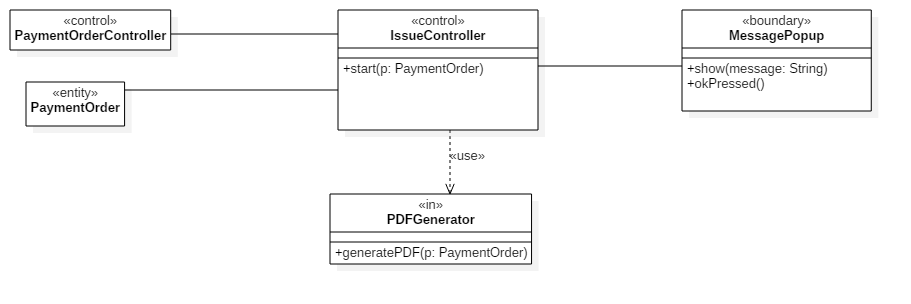
### Failed Sending



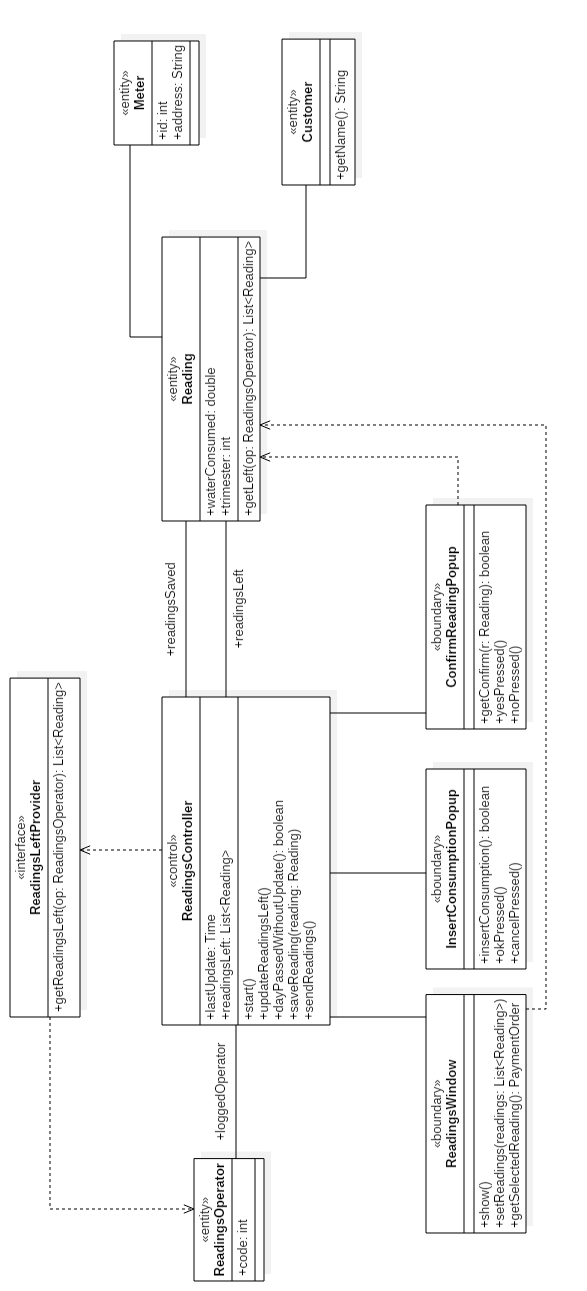
## Entity-Boundary-Control Class Diagram



### Desktop application controllers

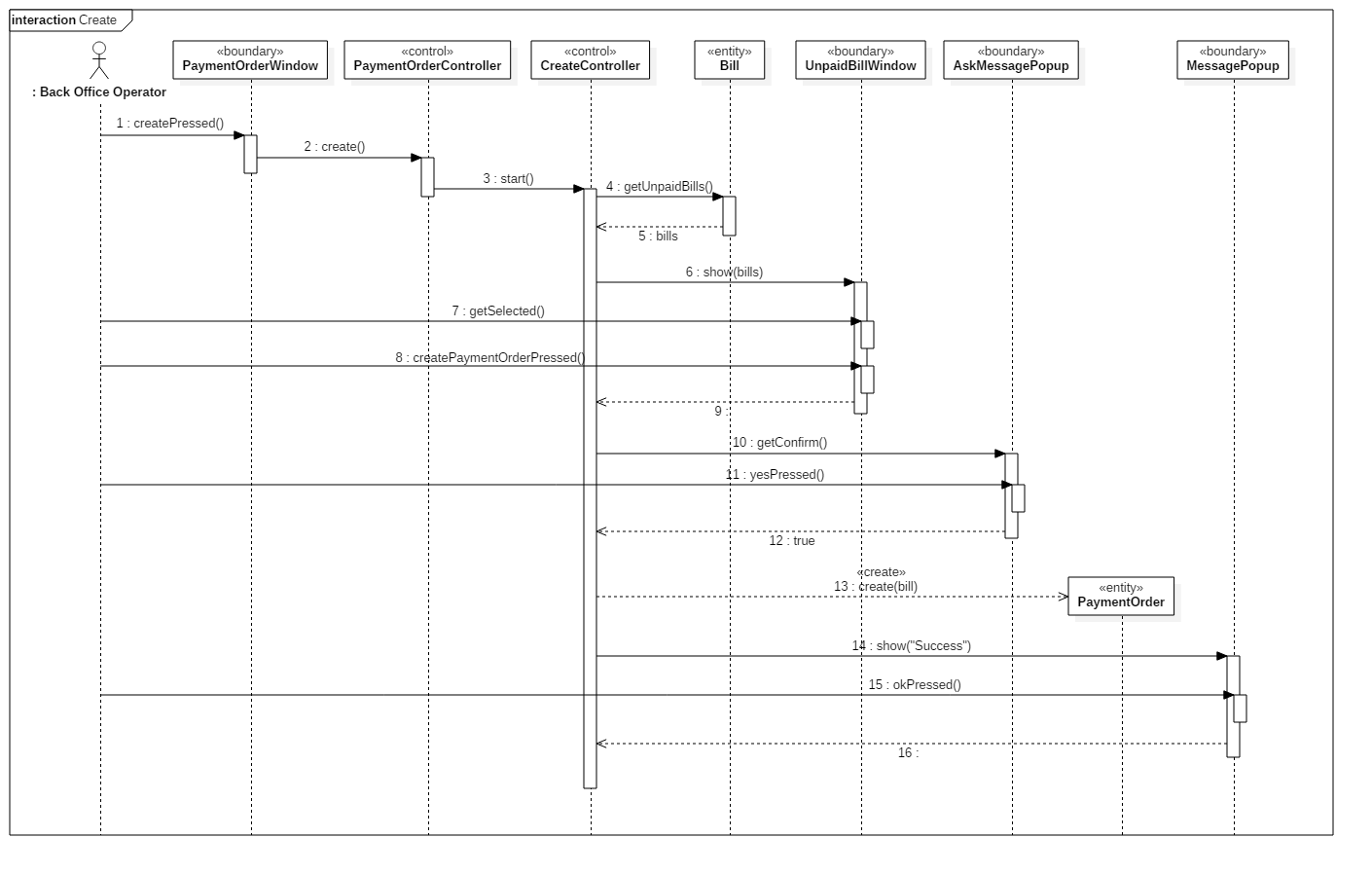


### Mobile application controller

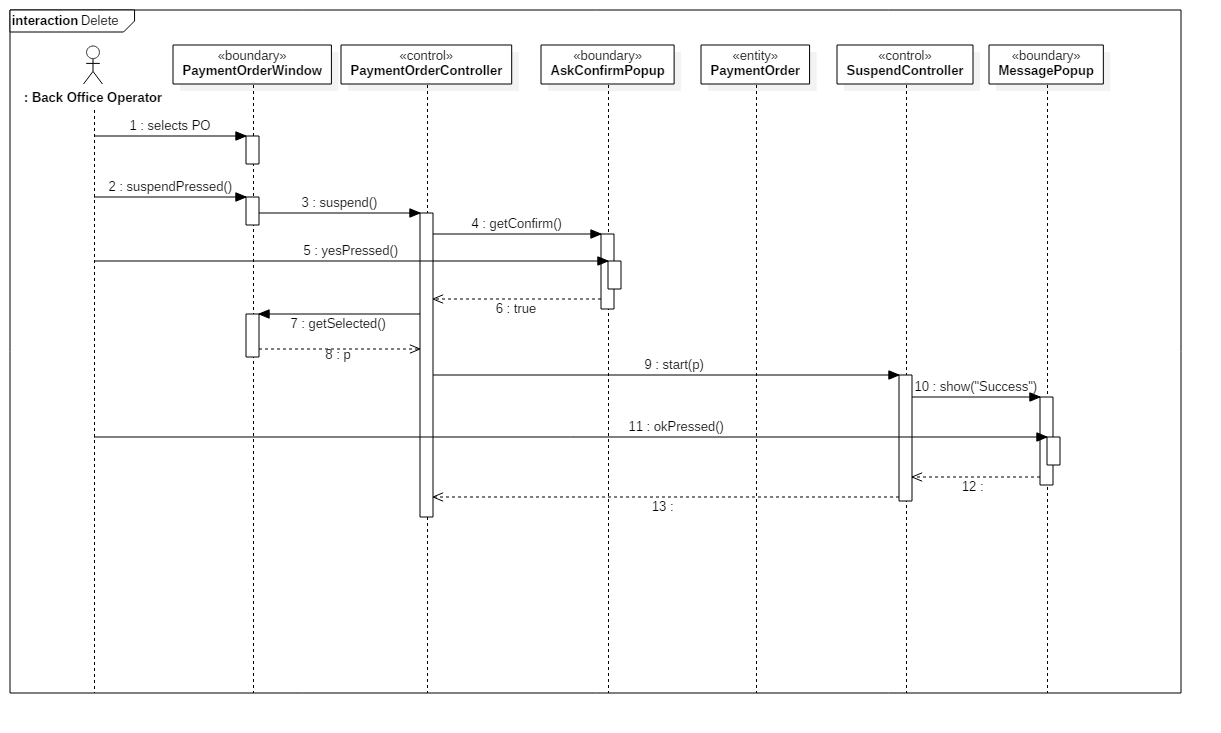


## Sequence Diagrams

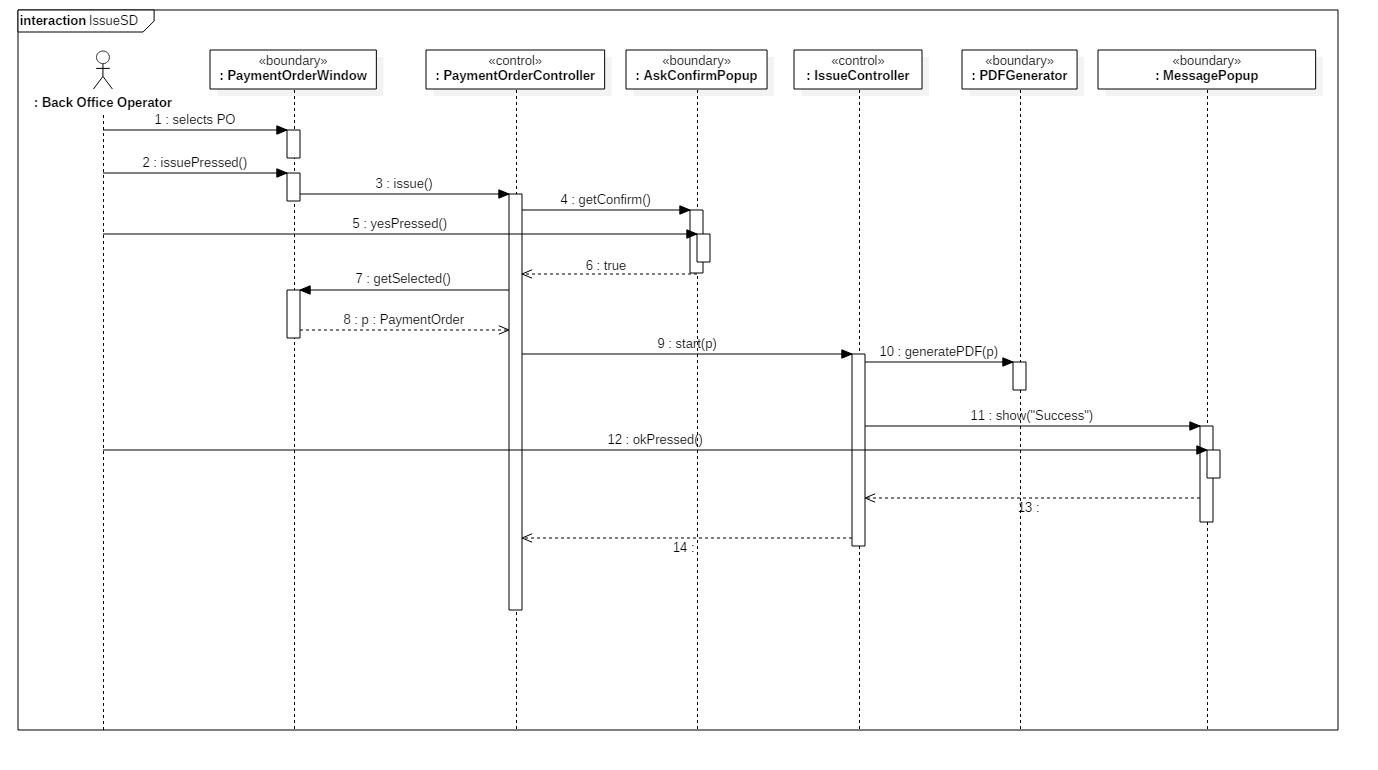
### Create payment order



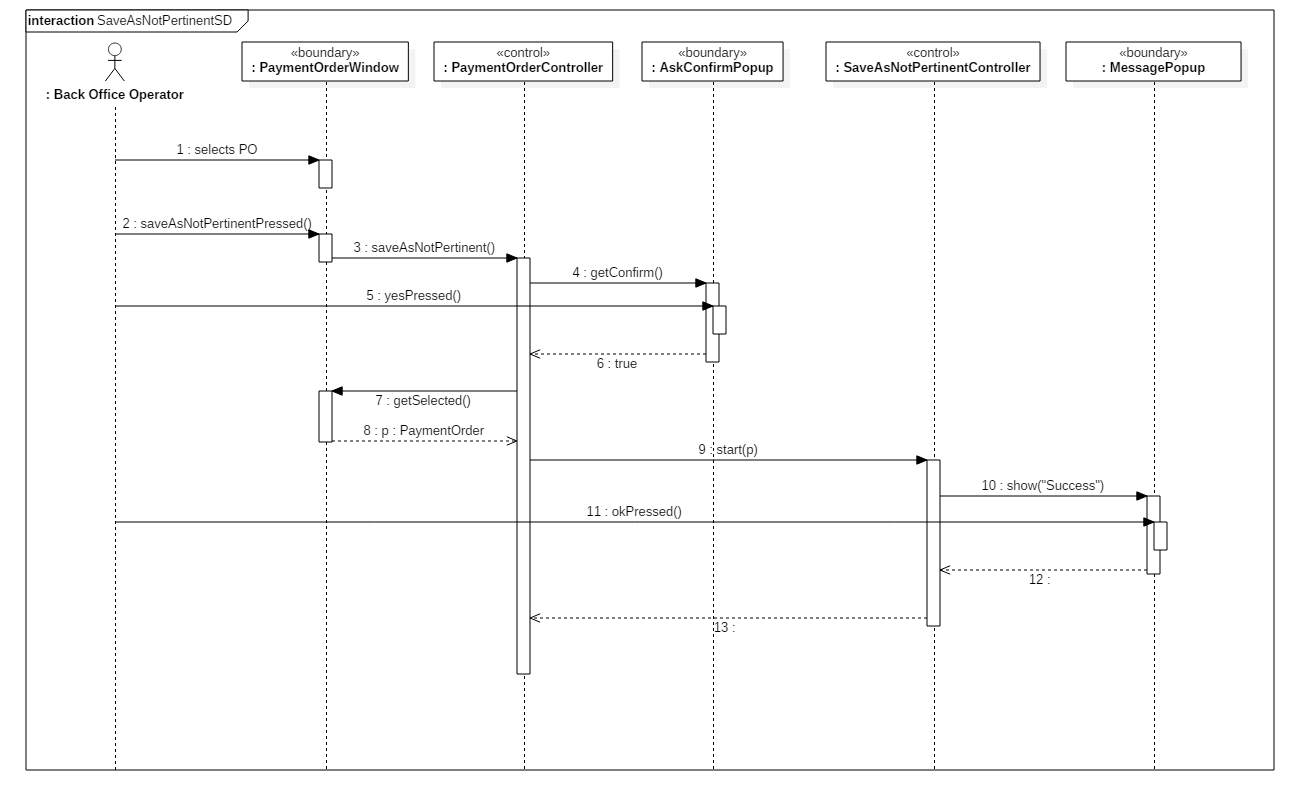
### Delete payment order



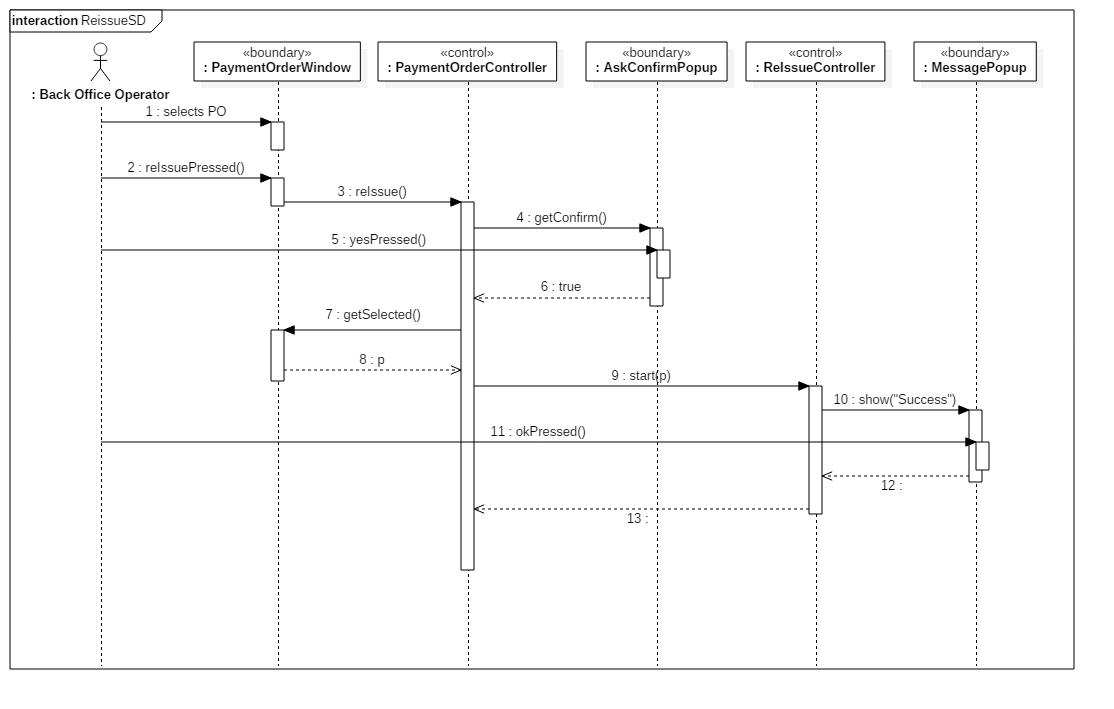
### Issue payment order



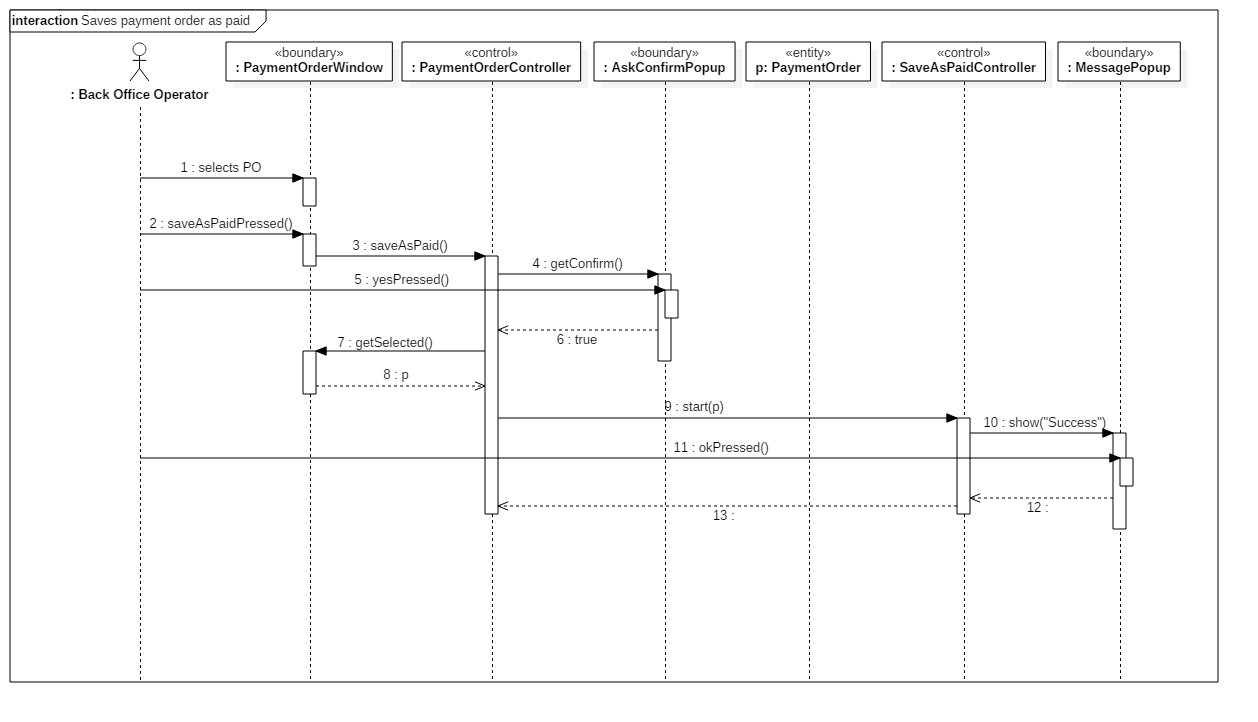
### Save payment order as not pertinent



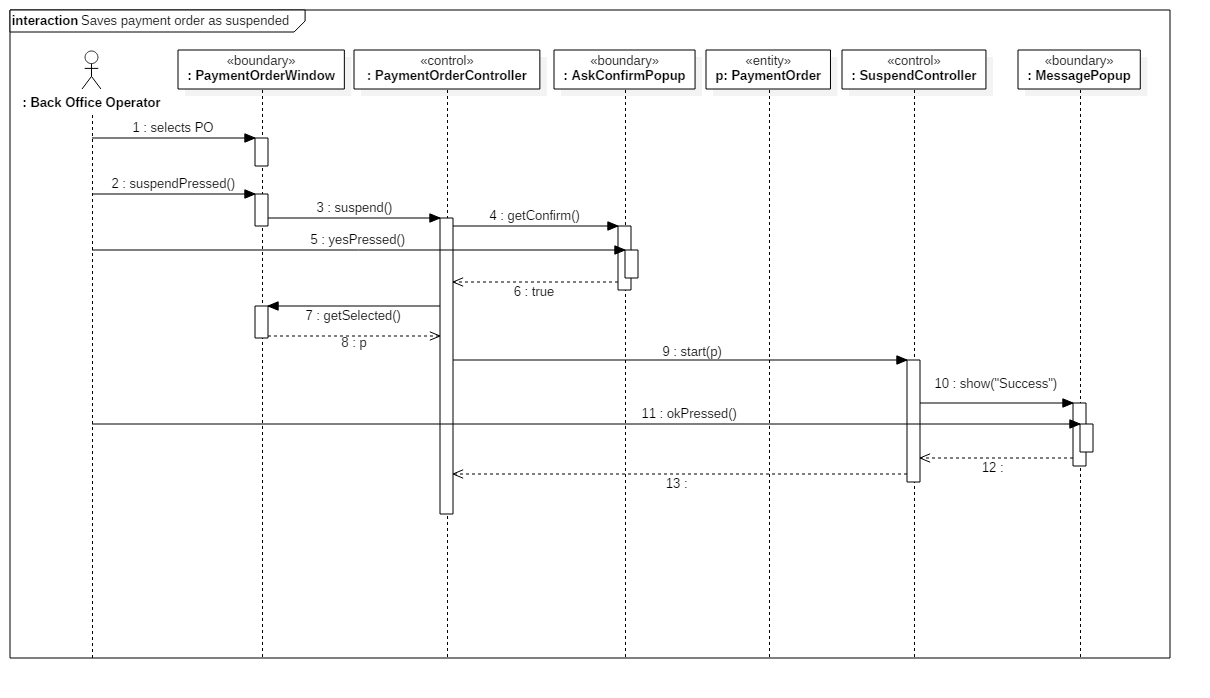
### Reissue payment order



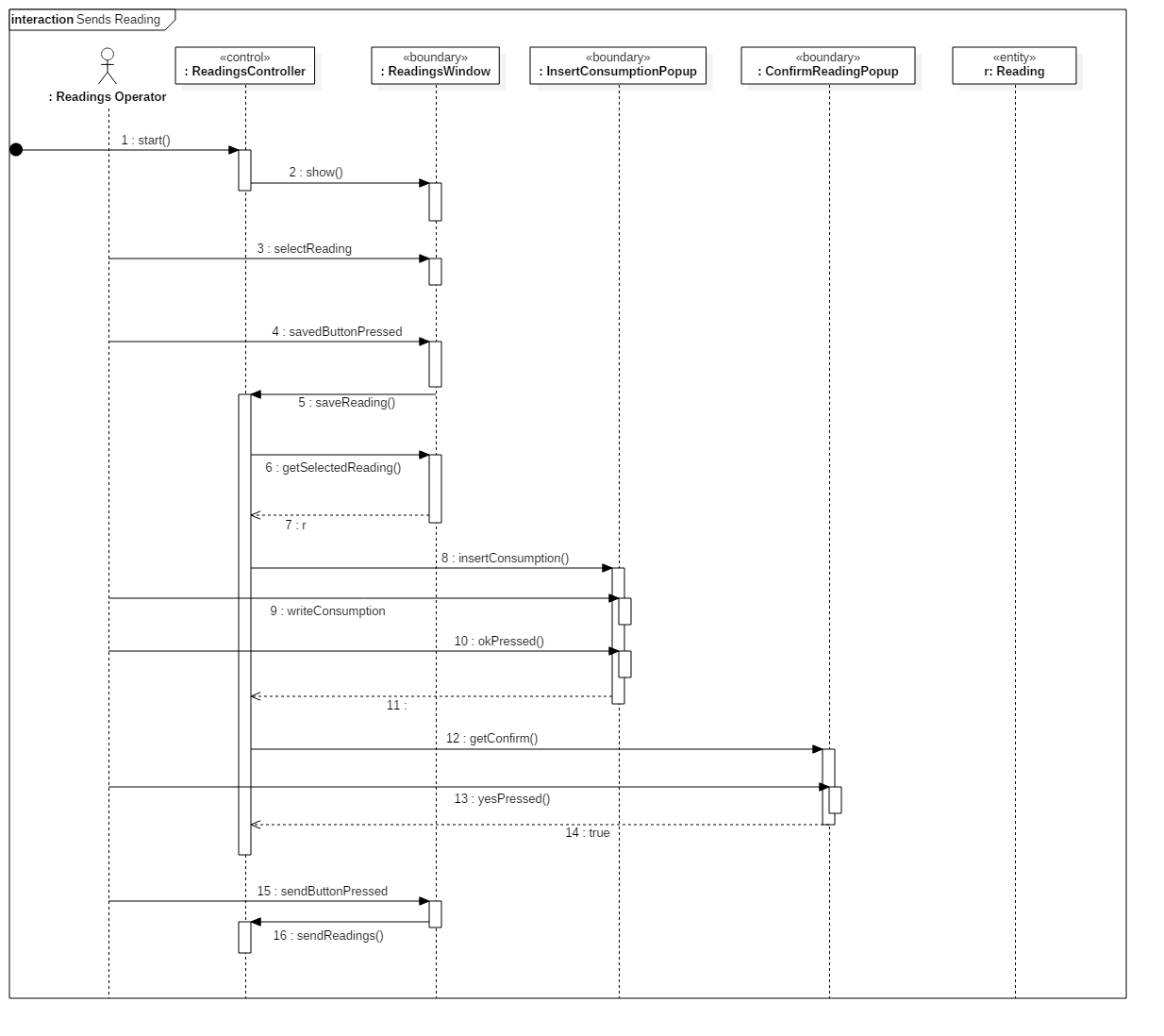
### Save payment order as paid



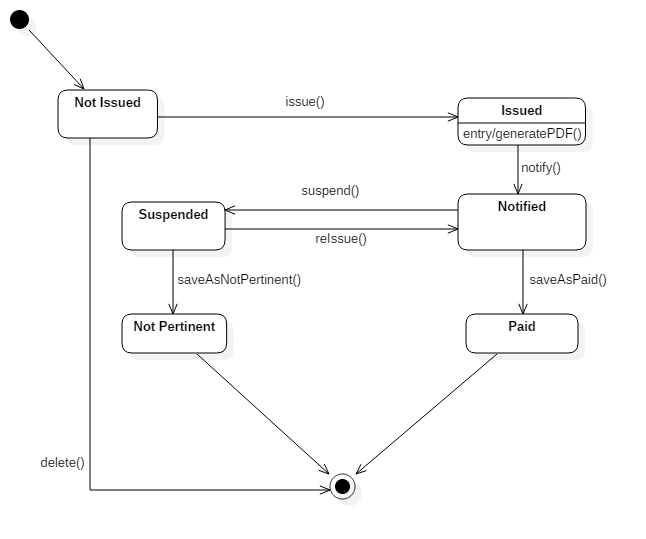
### Save payment order as suspended



### Send readings



## Payment Order Statechart Diagram

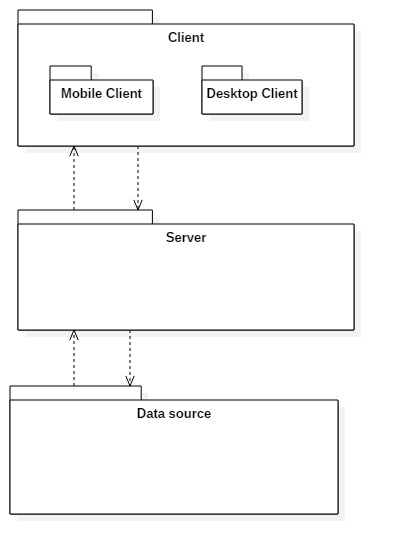


# Design documentation

## System design

### Software architecture

The system division is based on the Three-Tier Architecture, which divides the system into three modules: client, server, and data source.  
In this way most of the business logic is located in a central server, the code is more scalable and maintainable and the client is independent of the data source.



**Client**:  
Consists on the application which directly interacts with the users.   
This module is divided into two parts: a mobile application for readings operator, which is the mobile app that lets the users read the consumptions from the assigned meters, and a desktop application for back office operators.

**Server**:  
Represents the application's logical component, which provides authentication tokens, performs and verifies the correctness of the required operations, for example the status of a payment order.  
It was decided to use only one server to handle application requests, because it is estimated that the number of simultaneous connections is not high.

**Data source**:   
Relational DBMS in which persistent data are saved.

### Client server communication

Client and server communicate using HTTP messages. The server provides a session cookie after a successful login request and the client must send back the cookie for each further communication. The server listens on port 8081 and replies to client requests with the following response codes:

|  |  |  |
| --- | --- | --- |
| Response code | Reason phrase | Description |
| 200 | OK | The request has been successfully accomplished. |
| 500 | Internal server error | The request could not be accomplished due to an error encountered in the server. |
| 461 | Wrong id or password | The user has submitted a wrong id or password. |
| 462 | No session | The client has not sent a valid session cookie or the cookie sent has expired. |
| 463 | Missing parameter | The request expected a parameter which has not been sent. |
| 464 | Bad parameter values | One or more parameters presented an unexpected value. |
| 465 | Not practicable action | The requested action could not be applied to the given parameter. |

## Technologies

|  |  |  |
| --- | --- | --- |
| Technology | Version | Usages |
| Java | JDK 1.8 | Code language for all applications. |
| Apache Tomcat [1] | 8.5 | Server. |
| Java EE | Java EE 6 | Servlet for server web services. |
| JSON [2] |  | Client and server data exchange. |
| Oracle Database | 11g Express Edition | Database. |
| Android | SDK 16 – API 4.1 JDK 1.7 | Mobile application for Readings Operators. |
| Netbeans IDE | 8.2 | Code development for server and desktop applications. |
| Android Studio | 2.3.3 | Code development for Android environment. |

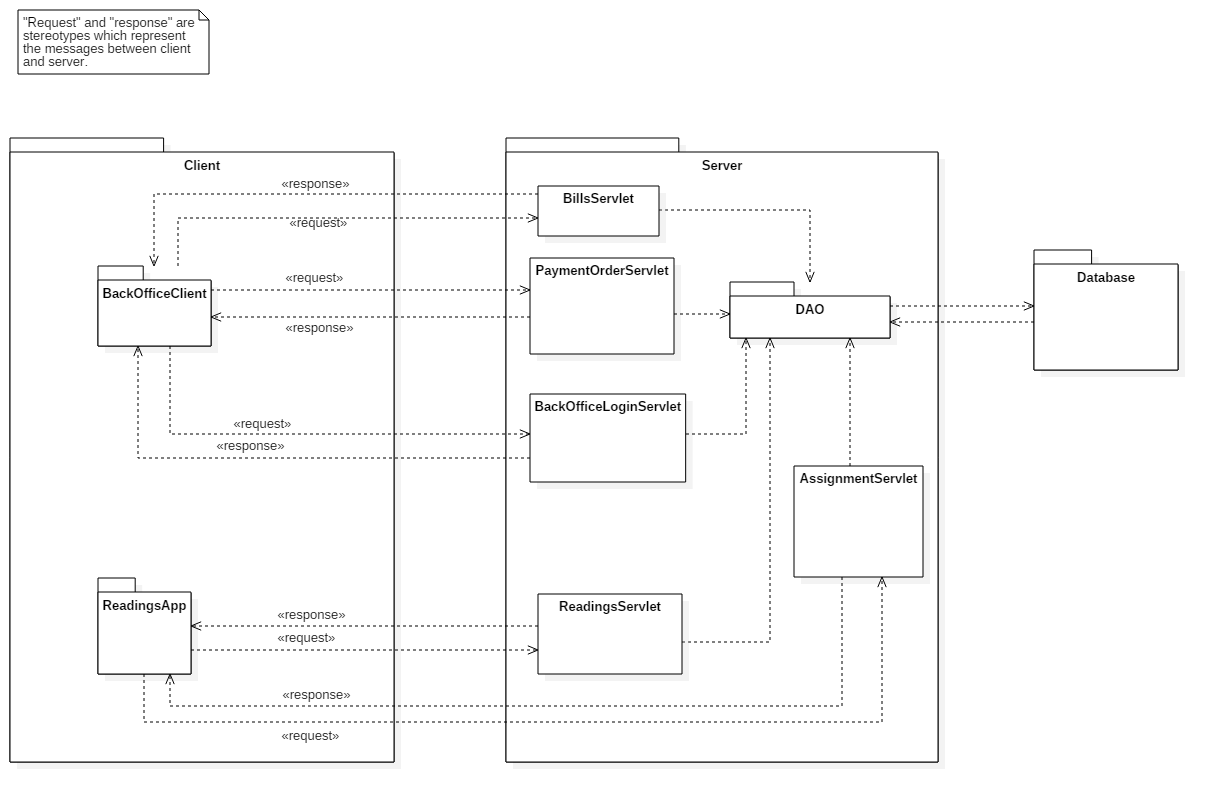
### Frameworks and Libraries

|  |  |  |
| --- | --- | --- |
| Name | Usages | References |
| Gson | Convert an object to the corresponding JSON string and vice versa | <https://github.com/google/gson/blob/master/UserGuide.md> |
| iText | Create and manipulate PDF files in Java | <https://developers.itextpdf.com/apis> |
| Oracle JDBC | Drivers for Oracle DBMS | <https://docs.oracle.com/cd/E11882_01/java.112/e16548/toc.htm> |
| Mockito | Create stub classes for testing | <http://site.mockito.org/> |
| jUnit 4.9 | Unit testing | <http://junit.org/junit4/> |

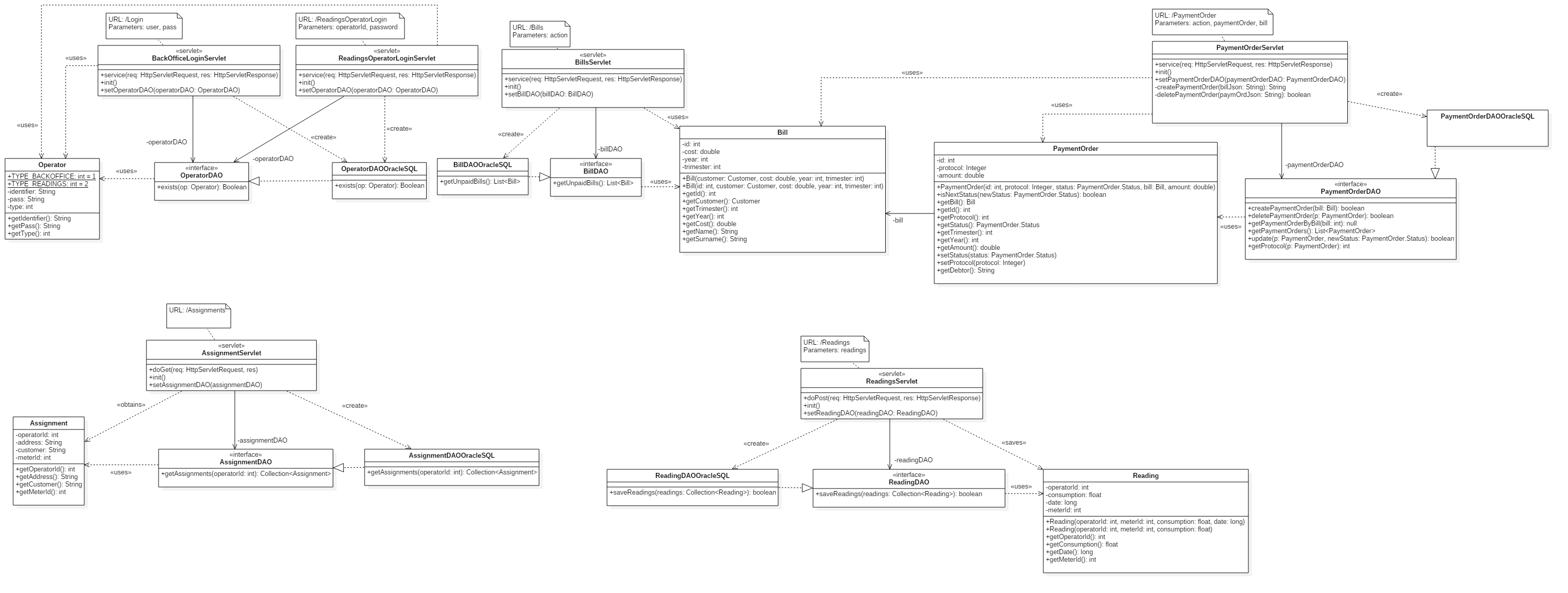
## Object design

### Class diagrams

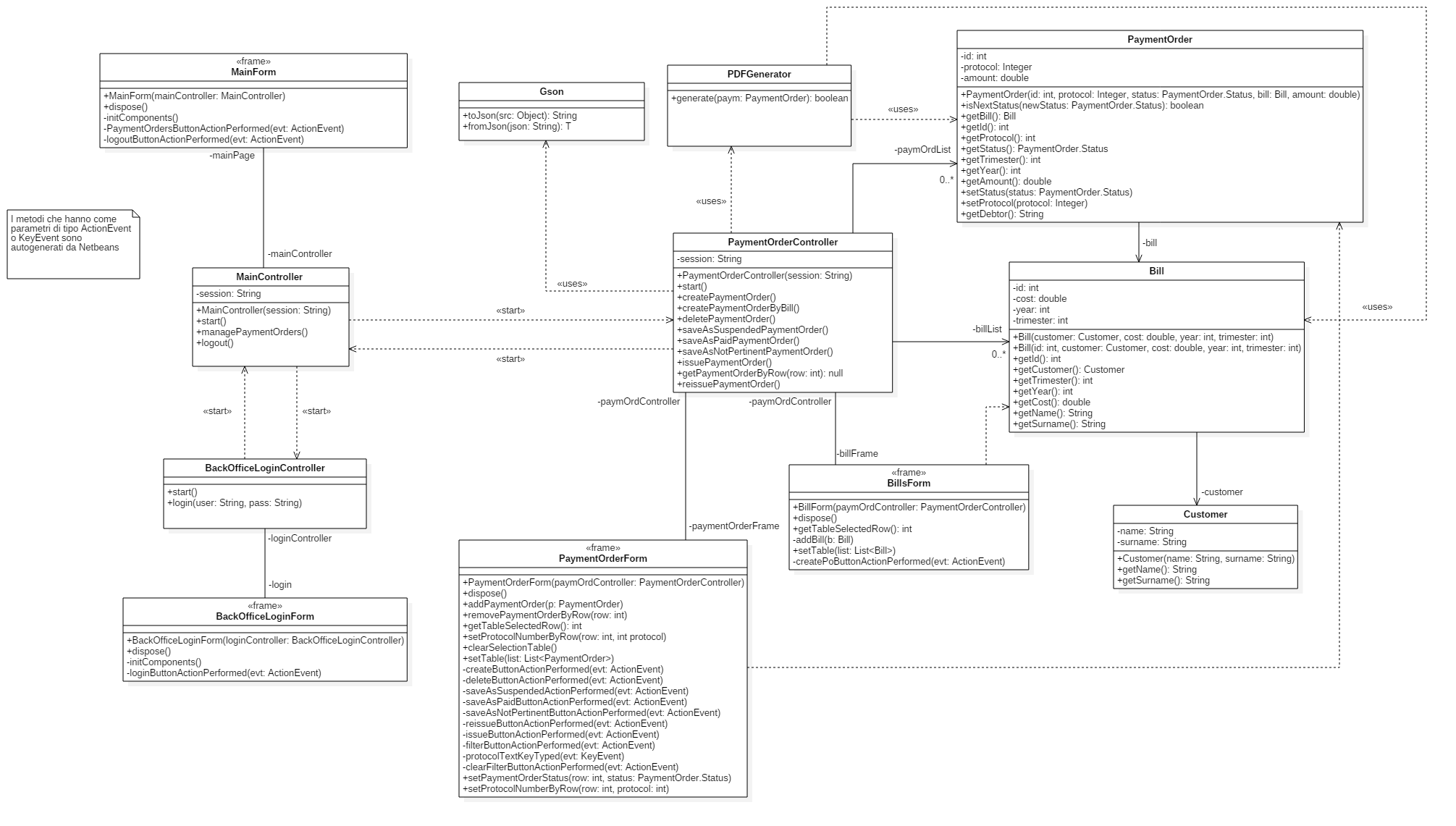
#### System



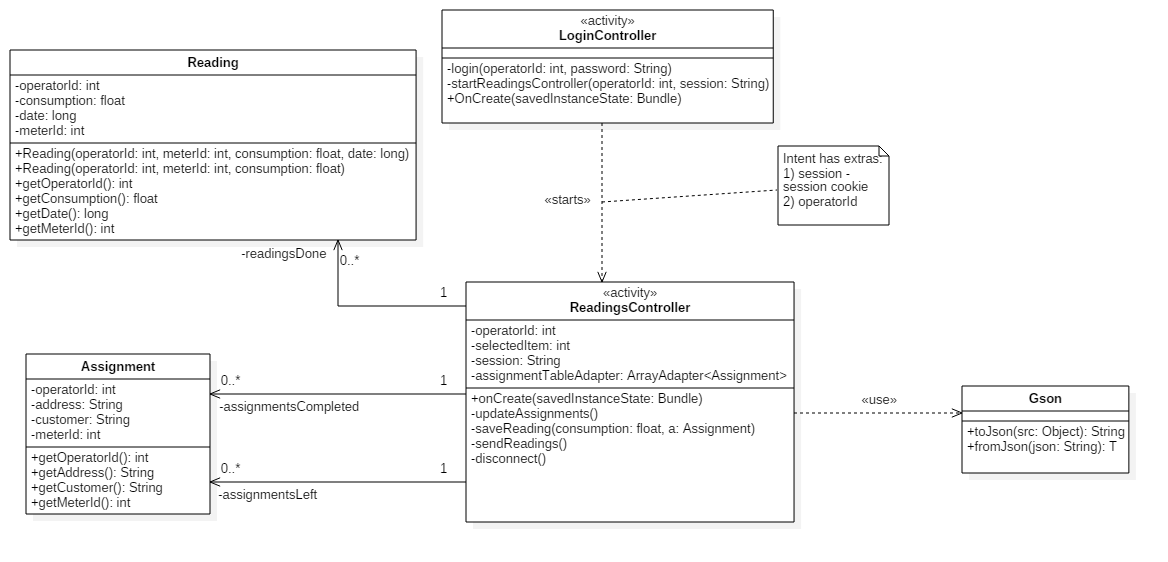
#### Server



#### Back-Office application

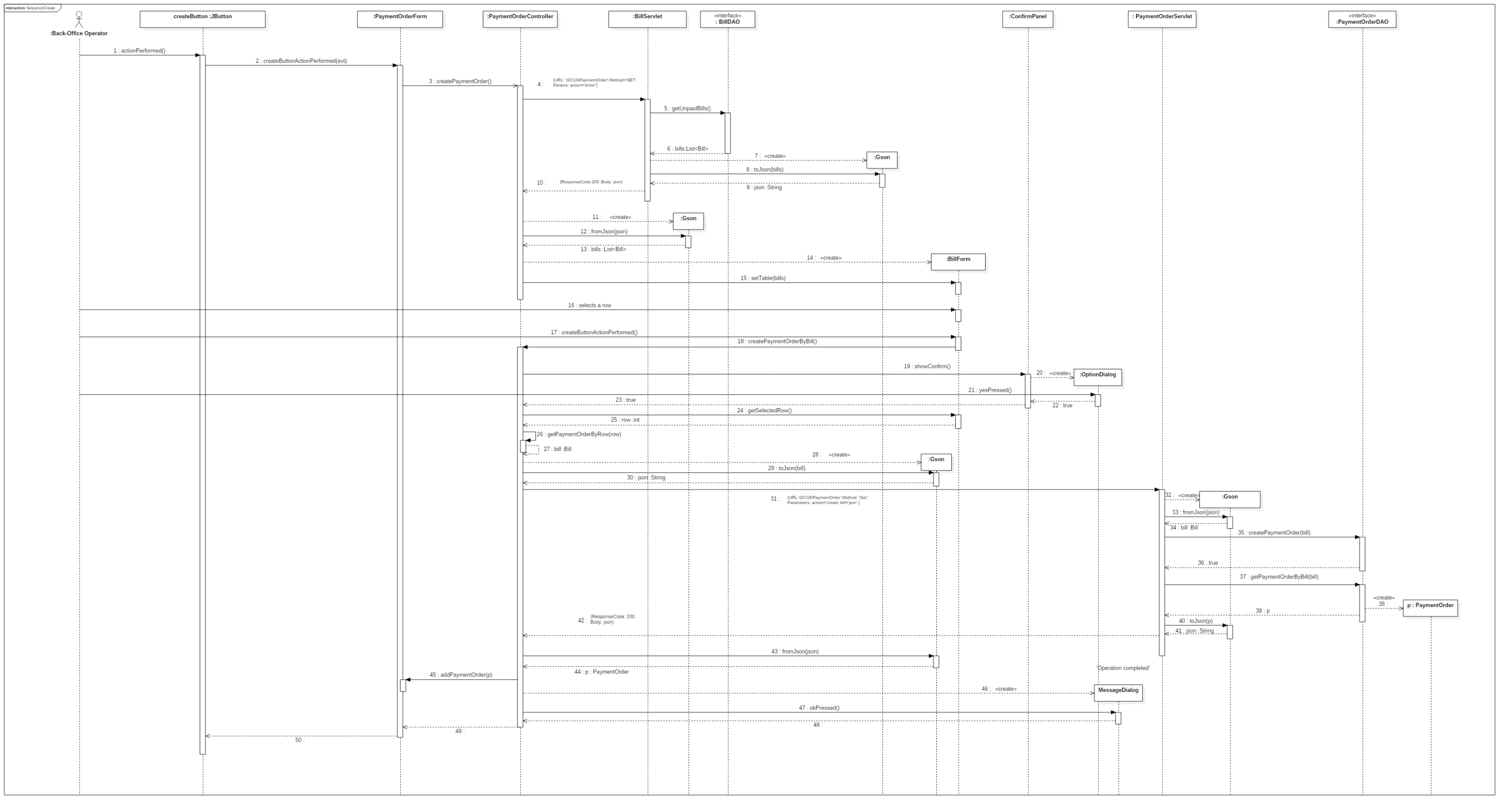


#### Mobile application



### Sequence diagrams

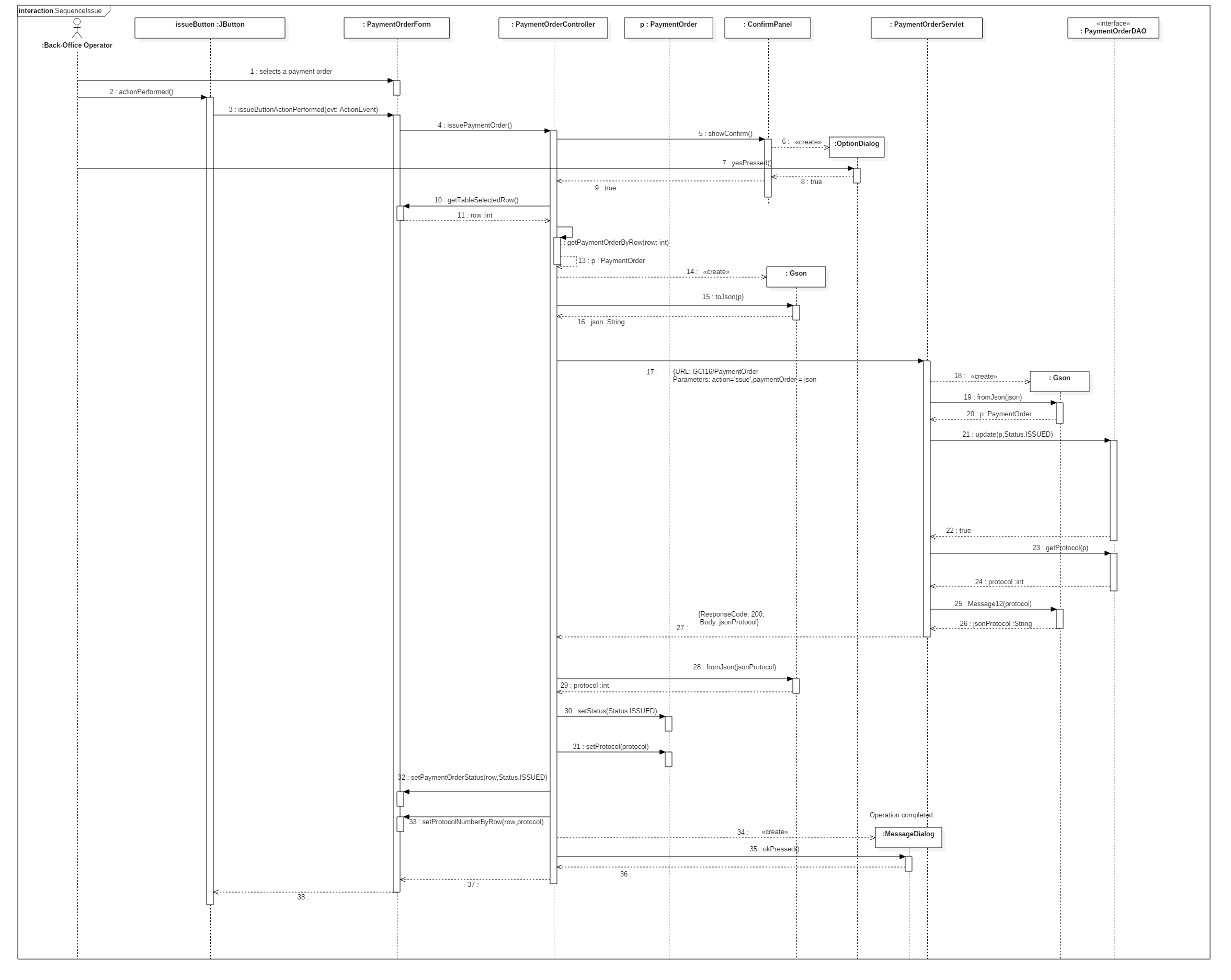
#### Create payment order



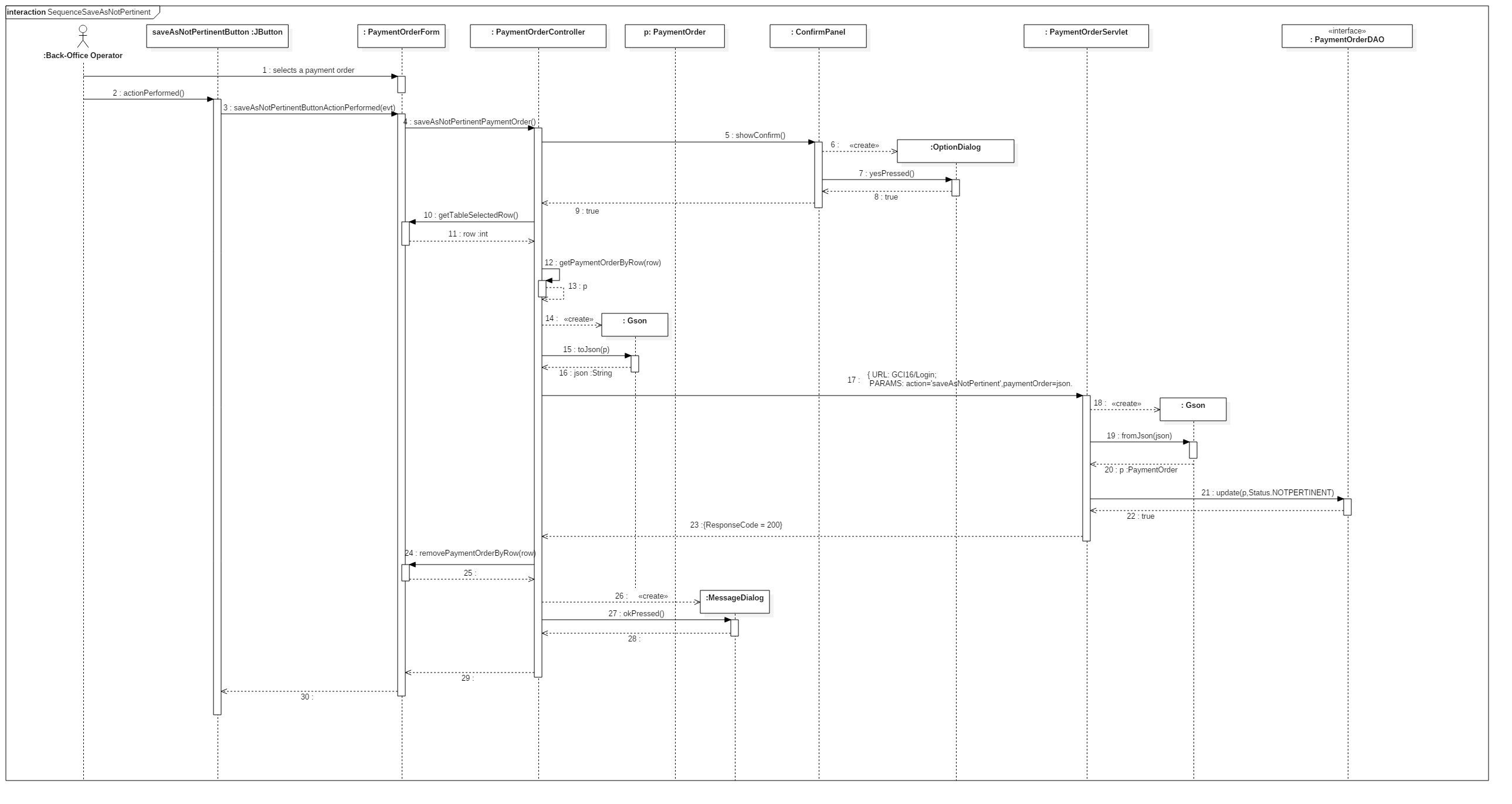
#### Delete payment order



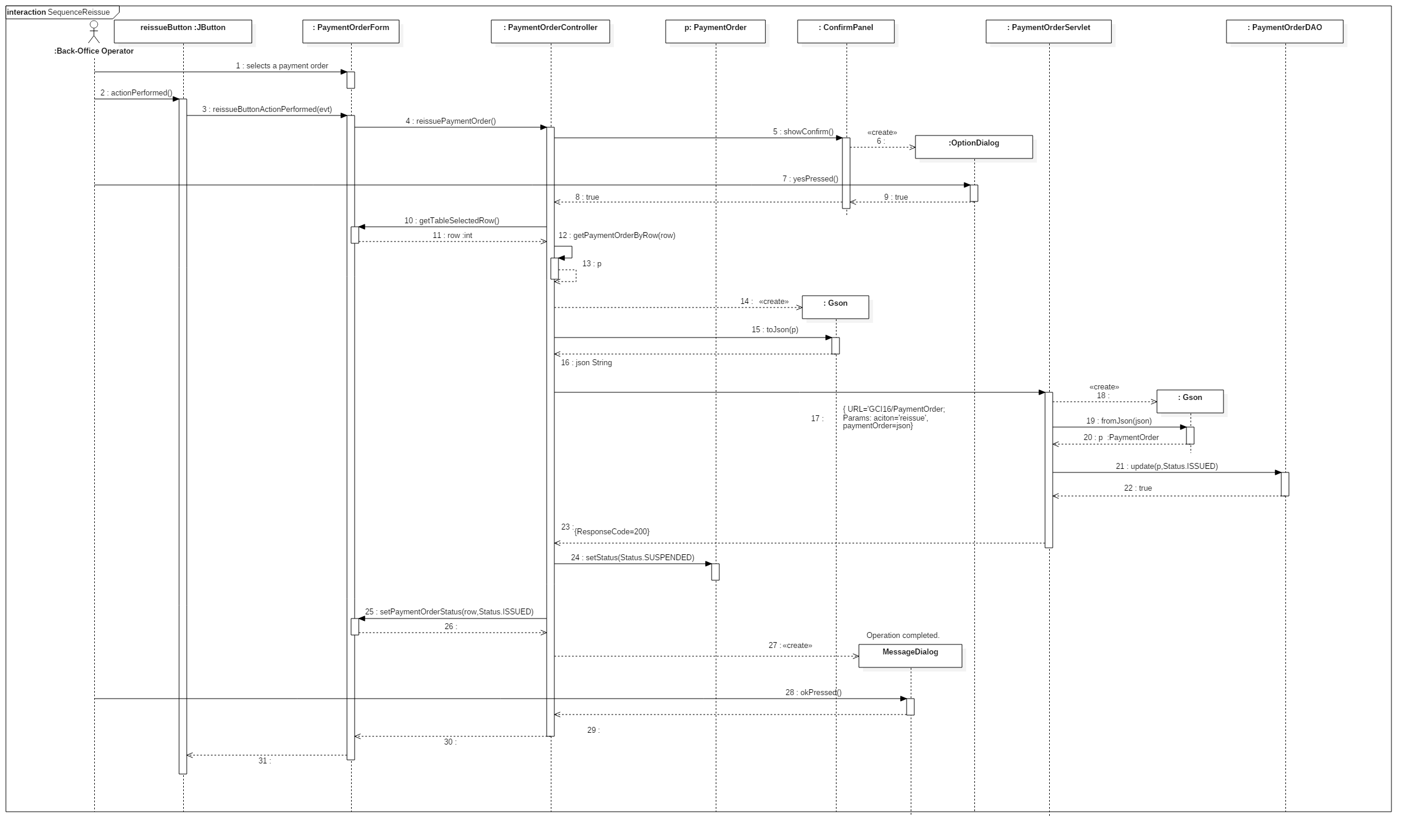
#### Issue payment order



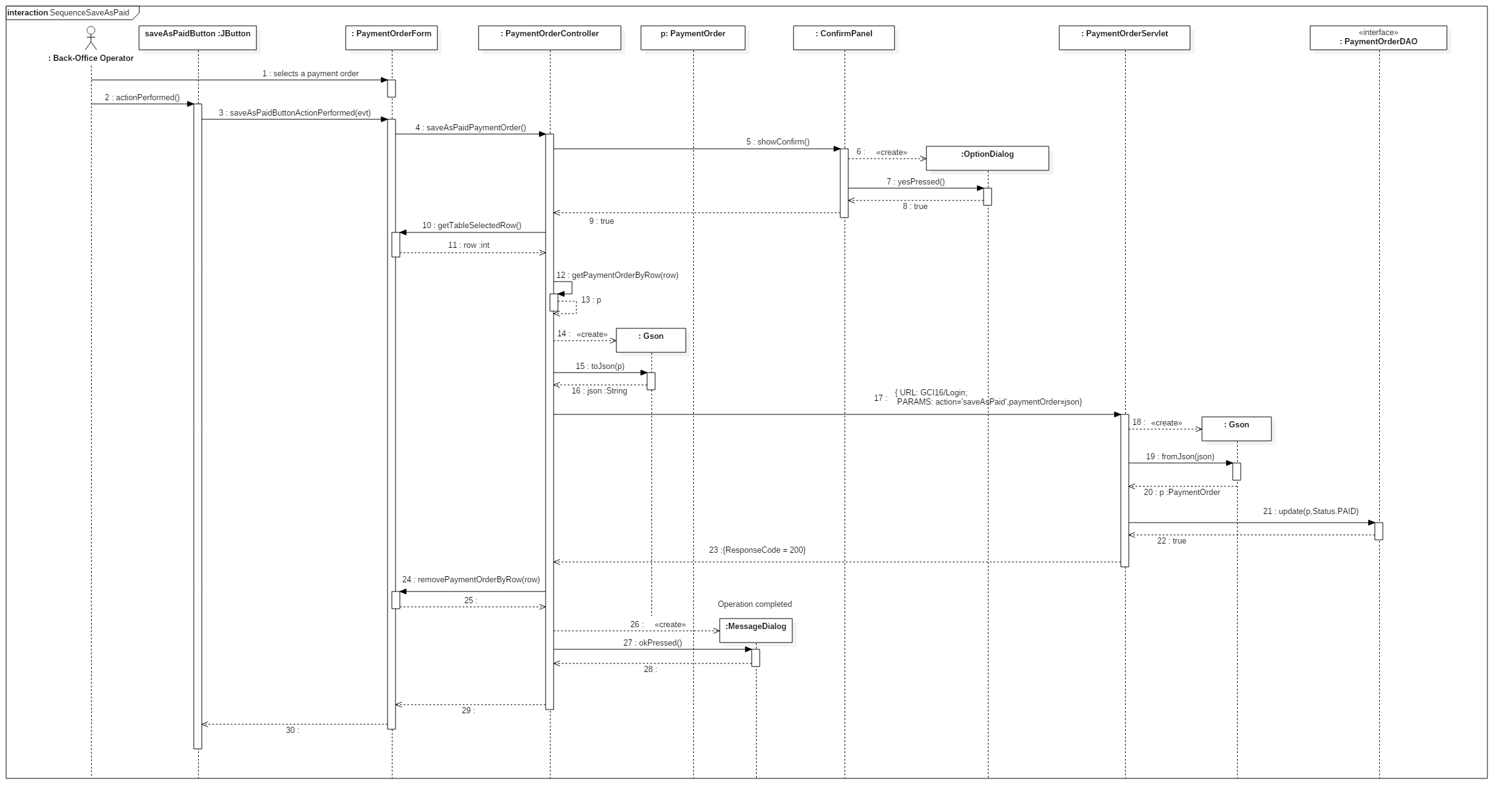
#### Save as not pertinent payment order



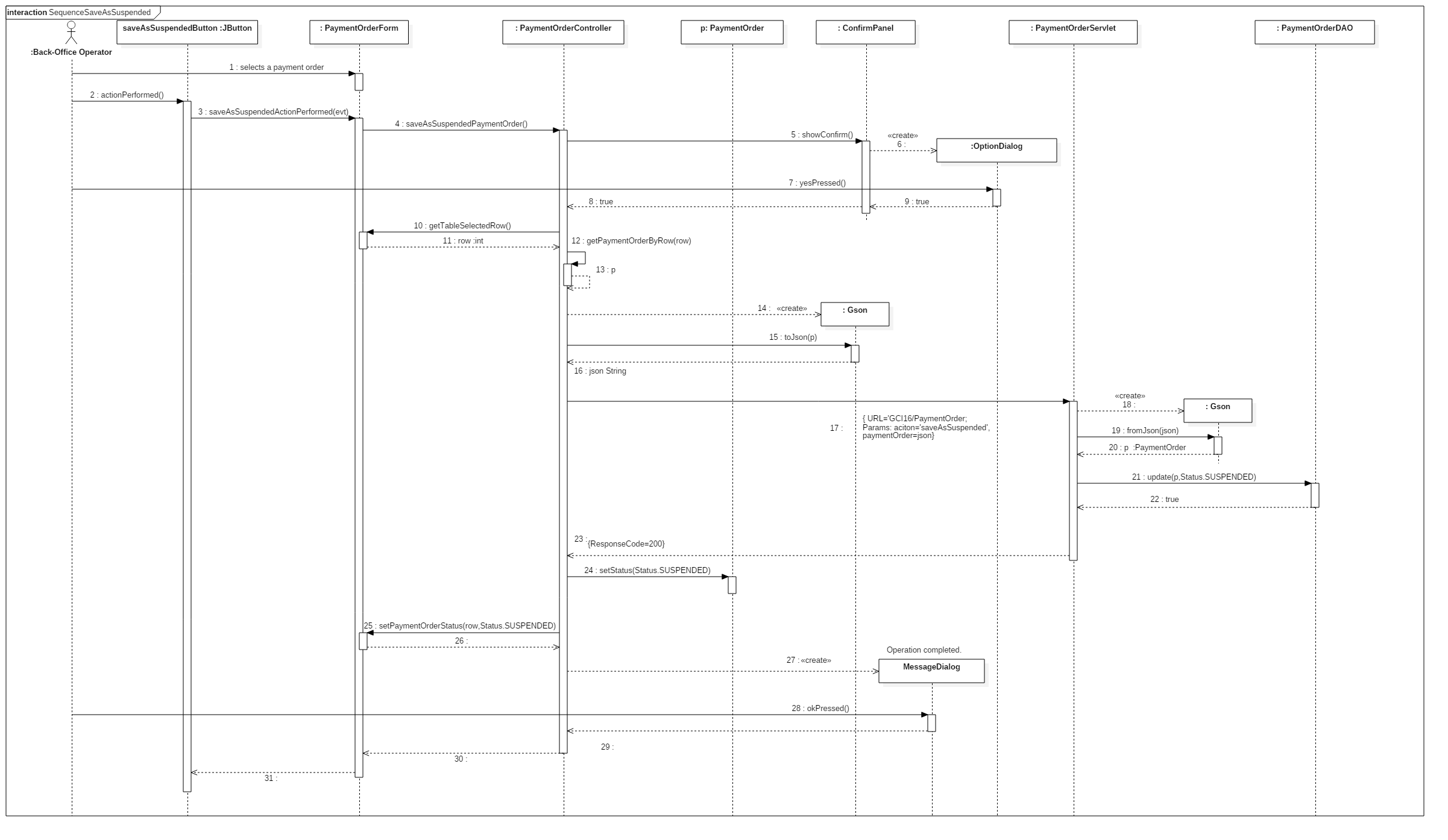
#### Reissue payment order



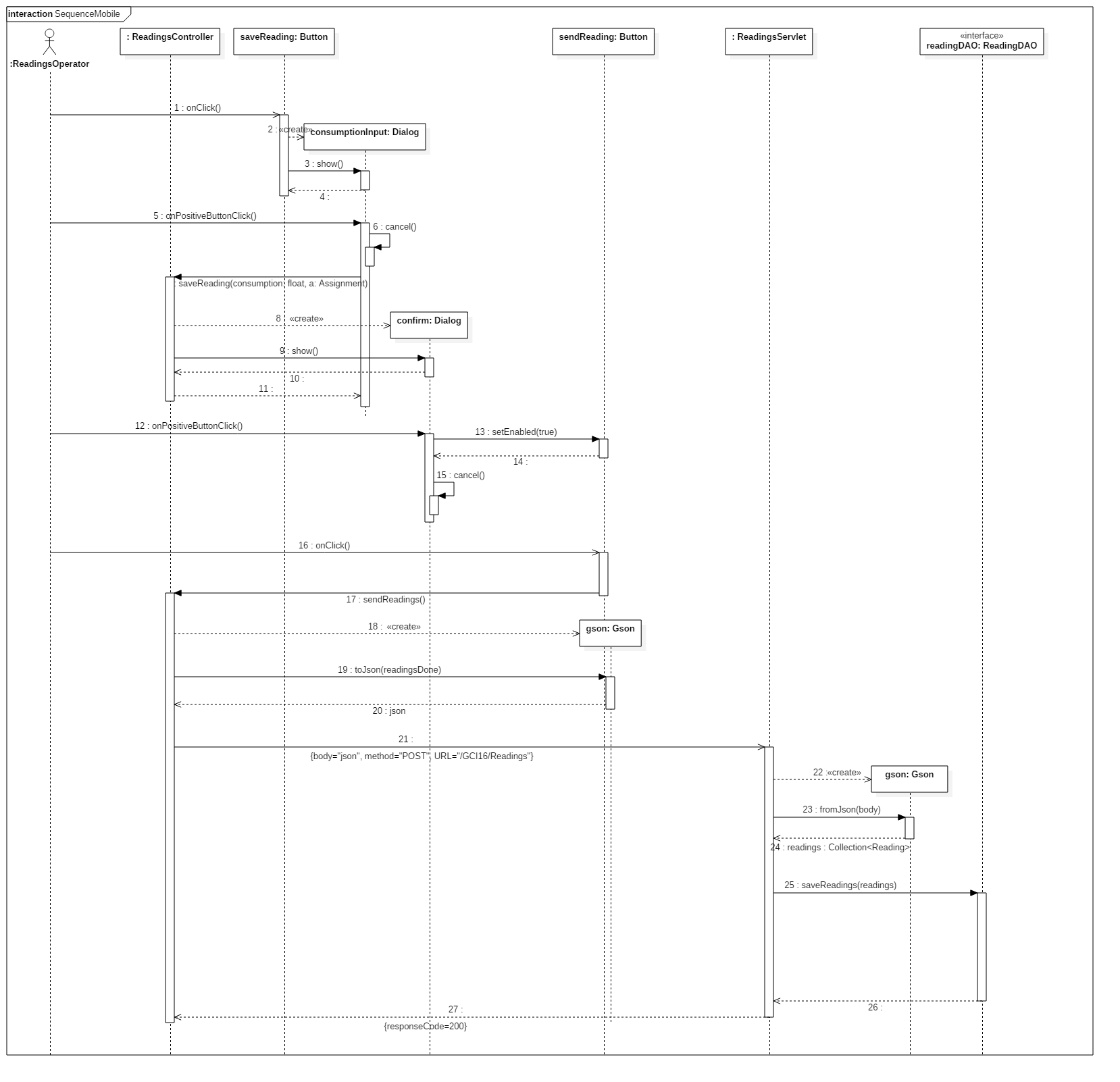
#### Save as paid payment order



#### Save as suspended payment order



#### Send readings



## CRC cards

### Server

|  |  |  |
| --- | --- | --- |
| **Class Name** | BackOfficeLoginServlet | |
| **Superclass** | HttpServlet | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Provide an access token for a Back-Office operator who submits correct credentials | | OperatorDAO, Operator |

|  |  |  |
| --- | --- | --- |
| **Class Name** | BillDAO | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Retrieve the bill unpaid for three months from the data source. | | Bill, Database |

|  |  |  |
| --- | --- | --- |
| **Class Name** | BillServlet | |
| **Superclass** | HttpServlet | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Provide an operator with the list of all the bills not paid for three months. | | BillDAO, Bill |

|  |  |  |
| --- | --- | --- |
| **Class Name** | PaymentOrderDAO | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Create a new payment order in the data source. | | PaymentOrder, Database |
| Delete a payment order from the data source. | | PaymentOrder, Database |
| Update the status of a payment order in the data source. | | PaymentOrder, Database |
| Retrieve payment orders data from the data source. | | PaymentOrder, Database |

|  |  |  |
| --- | --- | --- |
| **Class Name** | PaymentOrderServlet | |
| **Superclass** | HttpServlet | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Create a new payment order in the data source. | | PaymentOrderDAO, PaymentOrder |
| Delete a payment order from the data source. | | PaymentOrderDAO, PaymentOrder |
| Save a payment order as suspended in the data source. | | PaymentOrderDAO, PaymentOrder |
| Save a payment order as paid in the data source. | | PaymentOrderDAO, PaymentOrder |
| Save a payment order as not pertinent in the data source. | | PaymentOrderDAO, PaymentOrder |
| Issue a payment order in the data source. | | PaymentOrderDAO, PaymentOrder |
| Reissue a suspended payment order in the source. | | PaymentOrderDAO, PaymentOrder |

### Back-Office Application

|  |  |  |
| --- | --- | --- |
| **Class Name** | BackOfficeLoginController | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Let a Back-Office operator log in the system. | | BackOfficeLoginForm, BackOfficeLoginServlet |

|  |  |  |
| --- | --- | --- |
| **Class Name** | Bill | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Represent the bill for a trimester. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | BillForm | |
| **Superclass** | JFrame | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Show a table in which each row represents an unpaid bill. | | PaymentOrderController |
| Allow to select a bill from the bill’s table, of which a payment order can now be created. | |  |
| Give the row selected number of the bills’ table. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | Customer | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Represent a customer of the society. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | MainController | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Address the user to the macro-functionalities of the system. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | PaymentOrder | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Represent a payment order due to a not payed bill. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | PaymentOrderController | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Create a new payment order. | | PaymentOrderForm, PaymentOrder, BillForm, Bill, PaymentOrderServlet |
| Delete a payment order. | | PaymentOrderForm, PaymentOrder, PaymentOrderServlet |
| Save a payment order as suspended. | | PaymentOrderForm, PaymentOrder, PaymentOrderServlet |
| Save a payment order as paid. | | PaymentOrderForm, PaymentOrder, PaymentOrderServlet |
| Save a payment order as not pertinent. | | PaymentOrderForm, PaymentOrder, PaymentOrderServlet |
| Issue a payment order. | | PaymentOrderForm, PaymentOrder, PDFGenerator, PaymentOrderServlet |
| Reissue a suspended payment order. | | PaymentOrderForm, PaymentOrder, PaymentOrderServlet |

|  |  |  |
| --- | --- | --- |
| **Class Name** | PaymentOrderForm | |
| **Superclass** | JFrame | |
| **Subclasses** |  | |
| **Responsabilities** | | **Collaborators** |
| Show payment orders’ table in which each row represents a payment order. | | PaymentOrderController |
| Allow interaction with payment orders’ table. | |  |
| Give the row selected number of the payment orders’ table. | |  |
| Filter payment orders relatively to their attributes. | |  |

### Mobile Application

|  |  |  |
| --- | --- | --- |
| **Class Name** | Assignment | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsibilities** | | **Collaborators** |
| Represent the task of performing a reading of a meter. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | AssignmentServlet | |
| **Superclass** | HttpServlet | |
| **Subclasses** |  | |
| **Responsibilities** | | **Collaborators** |
| Provide an operator with the list of his assignments. | | Assignment, AssignmentDAO, Gson |

|  |  |  |
| --- | --- | --- |
| **Class Name** | LoginController | |
| **Superclass** | AppCompatActivity | |
| **Subclasses** |  | |
| **Responsibilities** | | **Collaborators** |
| Let the operator login from the mobile application. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | Reading | |
| **Superclass** |  | |
| **Subclasses** |  | |
| **Responsibilities** | | **Collaborators** |
| Represent the reading of the water consumption reported on a meter. | |  |

|  |  |  |
| --- | --- | --- |
| **Class Name** | ReadingsController | |
| **Superclass** | AppCompatActivity | |
| **Subclasses** |  | |
| **Responsibilities** | | **Collaborators** |
| Let the user save the reading consumption reported on a meter. | | Assignment, Reading, Gson |
| Update user’s list of assignments. | | Assignment, Gson |
| Send the readings done to a remote server. | | Reading, Gson |

|  |  |  |
| --- | --- | --- |
| **Class Name** | ReadingsServlet | |
| **Superclass** | HttpServlet | |
| **Subclasses** |  | |
| **Responsibilities** | | **Collaborators** |
| Save into the database the readings performed by a Readings Operator. | | Reading, ReadingDAO, Gson |

# Testing documentation

## System Testing

As agreed with the customer we have created a System Testing plan that tests each User Interface behaviour and each Use Case defined in the Requirements section.

The plan has been created using **WECT** (Weak Equivalence Class Testing).

### Back-Office Application

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | 1 | | | |
| **Test Name** | Back Office Login | | | |
| **Test Description** | Tests the Back Office application’s login form validity. | | | |
| **Input** | | **System status** | **Oracle** | **Output** |
| The user enters correct username and password | | The server is reachable | The application shows the main screen of the application |  |
| The user enters a wrong combination of username and/or password | | The server is reachable | The application shows an error message telling that he entered bad credentials |  |
| The user leaves a blank username and/or password field | | The server is reachable | The application shows an error message telling that one of the two fields is empty |  |
| The user doesn’t leave empty fields | | The server is not reachable | The application shows an error message telling that the server is not available |  |
| The user presses the “Close window” button | |  | The application closes |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | 2 | | | |
| **Test Name** | Payment Order Management | | | |
| **Test Description** | Tests the validity of the screen that lets the user manage the payment orders. | | | |
| **Input** | | **System status** | **Oracle** | **Output** |
| The user clicks on a row whose status is “NOTIFIED” | |  | Only “Create”, “Save as suspended” and “Save as paid” button are enabled |  |
| The user clicks on a row whose status is “NOTISSUED” | |  | Only “Create”, “Issue” and “Delete” button are enabled |  |
| The user clicks on a row whose status is “SUSPENDED” | |  | Only “Create”, “Reissue” and “Save as not pertinent” button are enabled |  |
| The user presses “Save as suspended” button and confirms the operation | | The server is reachable and the operation is successful | The status of the previously selected row changes to “SUSPENDED” and all buttons (“Create” excluded) are disabled |  |
| The user presses “Save as paid” button and confirms the operation | | The server is reachable and the operation is successful | The previously selected row is removed and all buttons (“Create” excluded) are disabled |  |
| The user presses “Issue” button and confirms the operation | | The server is reachable and the operation is successful | The status of the previously selected row changes to “ISSUED” and all buttons (“Create” excluded) are disabled |  |
| The user presses “Delete” button and confirms the operation | | The server is reachable and the operation is successful | The previously selected row is removed and all buttons (“Create” excluded) are disabled |  |
| The user presses “Reissue” button and confirms the operation | | The server is reachable and the operation is successful | The status of the previously selected row changes to “ISSUED” and all buttons (“Create” excluded) are disabled |  |
| The user fills some filtering fields | |  | Only the rows that satisfy the filtering criteria are left shown in the table |  |
| The user clicks on a operation button(“Create” excluded) and confirms it | | The server is not reachable | The application shows an error message |  |
| The user clicks on a operation button(“Create” excluded) and confirms it | | The server is reachable but the access token has expired | The application shows an error message and then the login screen |  |
| The user presses the “Close window” button | |  | The application shows the home screen |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | 3 | | | |
| **Test Name** | Bill form | | | |
| **Test Description** | Tests the frame where the user can choose a bill to create a payment order | | | |
| **Input** | | **System status** | **Oracle** | **Output** |
| The user selects a row from the table | |  | The “Create payment order” button is enabled |  |
| The user presses “Create payment order” button | | The server is not reachable | The application shows an error message |  |
| The user presses “Create payment order” button | | The server is reachable and the server-side operation is successful | The application shows the Payment Order management frame with the new payment order |  |
| The user presses “Create payment order” button | | The server is reachable but the server-side operation is not successful | The application shows an error message |  |
| The user presses “Create payment order” button | | The server is reachable but the access token has expired | The application shows an error message and then shows the login screen |  |
| The user presses the “Close window” button | |  | The application shows the Payment Order management frame |  |

### Mobile Application

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | 4 | | | |
| **Test Name** | Readings Operator Login | | | |
| **Test Description** | Tests the Readings Operator mobile application’s login form validity | | | |
| **Input** | | **System status** | **Oracle** | **Output** |
| The user enters correct operator id and password | | The server is reachable | The application shows the main screen of the application |  |
| The user enters a wrong combination of operator id and/or password | | The server is reachable | The application shows an error message telling that he entered bad credentials |  |
| The user leaves a blank operator id and/or password field | |  | “Login” button is disabled |  |
| The user doesn’t leave empty fields | | The server is not reachable | The application shows an error message telling that the server is not available |  |
| The user presses the back button | |  | The application closes |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | 5 | | | |
| **Test Name** | Readings Operator application | | | |
| **Test Description** | Tests the behaviour of the mobile application for Readings Operators | | | |
| **Input** | | **System status** | **Oracle** | **Output** |
| The user selects an assignment from the table | |  | The “Save reading” button is enabled |  |
| The user presses the menu button and then “Logout” button, and confirms the operation | |  | The application shows the login screen. |  |
| The user presses “Update” button | | The server is reachable | The application shows in the table the assignment of the operator, except the ones already completed but not sended |  |
| User presses “Save reading” button, enters the consumption and presses “Save” | |  | The assignment previously selected is removed from the table and the “Send readings” button is enabled |  |
| User presses “Send readings” button | | The server is reachable and the server-side operation is successful | “Send readings” button is disabled |  |
| The user presses “Send readings” or “Update” button | | The server is not reachable | The application shows an error message |  |
| The user presses “Send readings” or “Update” button | | The server is reachable but the server-side operation fails | The application shows an error message |  |
| The user presses “Send readings” or “Update” button | | The server is reachable but the access token has expired | The application shows an error message and then the login screen |  |
| The user presses back button | |  | The application goes in background |  |

## Unit Testing

We have decided to test a servlet with jUnit with the following test cases using **SECT** (**Strong Equivalence Class Testing**) method, in particular we have tested the *doPost* method of *ReadingsServlet*, that handles the storing of readings.

To create the stub classes we have used Mockito frameworks, which allows to easily create a class or interface instance and to define method returns

### Parameter equivalence classes

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameters domain** | **Description** | **Equivalence classes** | **Values** |
| A |  | A1 | Not null |
|  |  | A2 | Null |
| B |  | B1 | Collections of Reading instances |
|  |  | B2 | Missing parameter in request |
|  |  | B3 | Well-formed JSON strings but not a collection of Reading instances |
|  |  | B4 | Malformed JSON strings |
| C |  | C1 | TRUE |
|  |  | C2 | FALSE |

### Test cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **A** | **B** | **C** | **Response code** |
| SE 1 | A1 | B1 | C1 | 200 |
| SE 2 | A1 | B1 | C2 | 500 |
| SE 3 | A1 | B2 | C1 | 463 |
| SE 4 | A1 | B2 | C2 | 463 |
| SE 5 | A1 | B3 | C1 | 464 |
| SE 6 | A1 | B3 | C2 | 464 |
| SE 7 | A1 | B4 | C1 | 464 |
| SE 8 | A1 | B4 | C2 | 464 |
| SE 9 | A2 | B1 | C1 | 461 |
| SE 10 | A2 | B1 | C2 | 461 |
| SE 11 | A2 | B2 | C1 | 461 |
| SE 12 | A2 | B2 | C2 | 461 |
| SE 13 | A2 | B3 | C1 | 461 |
| SE 14 | A2 | B3 | C2 | 461 |
| SE 15 | A2 | B4 | C1 | 461 |
| SE 16 | A2 | B4 | C2 | 461 |

### Code

import com.google.gson.Gson;

import dao.interfaces.ReadingDAO;

import entities.Customer;

import entities.Reading;

import java.io.IOException;

import java.util.LinkedList;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpServletResponseWrapper;

import javax.servlet.http.HttpSession;

import junit.framework.Assert;

import org.junit.Before;

import org.junit.Test;

import org.mockito.Mockito;

import server.mobile.ReadingsServlet;

/\*\*

\* Testing of doPost method using JUnit 4.9 in SECT method

\* @author GCI16\_25

\*/

public final class ReadingsServletTest extends Mockito{

private final ReadingsServlet servlet = new ReadingsServlet();

private HttpServletRequest request;

private HttpServletResponse response;

private ReadingDAO readingDAO;

private HttpSession session;

private static class HttpServletResponseForTest extends HttpServletResponseWrapper{

private int status = 200;

private String message;

public HttpServletResponseForTest() {

super(mock(HttpServletResponse.class));

}

@Override

public void setStatus(int status){

this.status = status;

}

@Override

public void sendError(int sc) throws IOException {

setStatus(sc);

}

@Override

public void sendError(int sc, String msg) throws IOException {

sendError(sc);

this.message = msg;

}

@Override

public int getStatus() {

return this.status;

}

}

@Before

public void prepareTest(){

//servlet = mock(ReadingsServlet.class);

request = mock(HttpServletRequest.class);

response = new HttpServletResponseForTest();

readingDAO = mock(ReadingDAO.class);

session = mock(HttpSession.class);

servlet.setReadingDAO(readingDAO);

}

/\* Test 1

Session found, JSON readings well-formed and DAO returns true \*/

@Test

public void test\_okSession\_okJSON\_trueDAO(){

Gson gson = new Gson();

Reading readingObject = new Reading(1, 1, 1);

LinkedList<Reading> readingCollection = new LinkedList<>();

readingCollection.add(readingObject);

String reading = gson.toJson(readingCollection);

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn(reading); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

try {

servlet.doPost(request, response);

} catch (IOException ex) {

Assert.assertTrue(false);

}

Assert.assertEquals(200, response.getStatus());

}

/\* Test 2

Session found, JSON readings well-formed and DAO returns false \*/

@Test

public void test\_okSession\_okJSON\_falseDAO() throws Exception{

Gson gson = new Gson();

Reading readingObject = new Reading(1, 1, 1);

LinkedList<Reading> readingCollection = new LinkedList<>();

readingCollection.add(readingObject);

String reading = gson.toJson(readingCollection);

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn(reading); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(500, response.getStatus());

}

/\* Test 3

Session found, no JSON readings and DAO returns true \*/

@Test

public void test\_okSession\_noJSON\_trueDAO() throws Exception{

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn(null); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(463, response.getStatus());

}

/\* Test 4

Session found, no JSON readings and DAO returns false \*/

@Test

public void test\_okSession\_noJSON\_falseDAO() throws Exception{

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn(null); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(463, response.getStatus());

}

/\* Test 5

Session found, not reading JSON and DAO returns true \*/

@Test

public void test\_okSession\_notReadingJSON\_trueDAO() throws Exception{

Gson gson = new Gson();

Customer customerObject = new Customer("carlo", "de vita");

LinkedList<Customer> customerCollection = new LinkedList<>();

customerCollection.add(customerObject);

String customer = gson.toJson(customerCollection);

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn(customer); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(464, response.getStatus());

}

/\* Test 6

Session found, not reading JSON and DAO returns false \*/

@Test

public void test\_okSession\_notReadingJSON\_falseDAO() throws Exception{

Gson gson = new Gson();

Customer customerObject = new Customer("carlo", "de vita");

LinkedList<Customer> customerCollection = new LinkedList<>();

customerCollection.add(customerObject);

String customer = gson.toJson(customerCollection);

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn(customer); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(464, response.getStatus());

}

/\* Test 7

Session found, JSON readings not well-formed and DAO returns true \*/

@Test

public void test\_okSession\_notWellFormedJSON\_trueDAO() throws Exception{

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn("Provaaa"); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(464, response.getStatus());

}

/\* Test 8

Session found, JSON readings not well-formed and DAO returns false \*/

@Test

public void test\_okSession\_notWellFormedJSON\_falseDAO() throws Exception{

when(request.getSession(false)).thenReturn(session); // Set stub session

when(request.getParameter("readings")).thenReturn("Provaaa"); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(464, response.getStatus());

}

/\* Test 9

Session does not exist, JSON readings well-formed and DAO returns true \*/

@Test

public void test\_noSession\_okJSON\_trueDAO() throws Exception{

Gson gson = new Gson();

Reading readingObject = new Reading(1, 1, 1);

LinkedList<Reading> readingCollection = new LinkedList<>();

readingCollection.add(readingObject);

String reading = gson.toJson(readingCollection);

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn(reading); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

/\* Test 10

Session does not exist, JSON readings well-formed and DAO returns false \*/

@Test

public void test\_noSession\_okJSON\_falseDAO() throws Exception{

Gson gson = new Gson();

Reading readingObject = new Reading(1, 1, 1);

LinkedList<Reading> readingCollection = new LinkedList<>();

readingCollection.add(readingObject);

String reading = gson.toJson(readingCollection);

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn(reading); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

/\* Test 11

Session does not exist, no JSON readings and DAO returns true \*/

@Test

public void test\_noSession\_noJSON\_trueDAO() throws Exception{

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn(null); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

/\* Test 12

Session does not exist, no JSON readings and DAO returns false \*/

@Test

public void test\_noSession\_noJSON\_falseDAO() throws Exception{

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn(null); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

/\* Test 13

Session does not exist, not reading JSON and DAO returns true \*/

@Test

public void test\_noSession\_notReadingJSON\_trueDAO() throws Exception{

Gson gson = new Gson();

Customer customerObject = new Customer("carlo", "de vita");

LinkedList<Customer> customerCollection = new LinkedList<>();

customerCollection.add(customerObject);

String customer = gson.toJson(customerCollection);

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn(customer); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

/\* Test 14

Session does not exist, not reading JSON and DAO returns false \*/

@Test

public void test\_noSession\_notReadingJSON\_falseDAO() throws Exception{

Gson gson = new Gson();

Customer customerObject = new Customer("carlo", "de vita");

LinkedList<Customer> customerCollection = new LinkedList<>();

customerCollection.add(customerObject);

String customer = gson.toJson(customerCollection);

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn(customer); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

/\* Test 15

Session does not exist, JSON readings not well-formed and DAO returns true \*/

@Test

public void test\_noSession\_notWellFormedJSON\_trueDAO() throws Exception{

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn("Provaaa"); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(true); // Set stub readings parameter

servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

/\* Test 16

Session does not exist, JSON readings not well-formed and DAO returns false \*/

@Test

public void test\_noSession\_notWellFormedJSON\_falseDAO() throws Exception{

when(request.getSession(false)).thenReturn(null); // Set stub session

when(request.getParameter("readings")).thenReturn("Provaaa"); // Set stub readings parameter

when(readingDAO.saveReadings(any())).thenReturn(false); // Set stub readings parameter

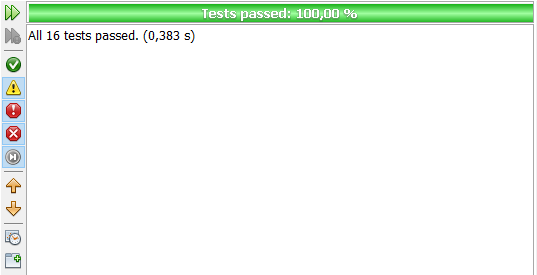
servlet.doPost(request, response);

Assert.assertEquals(462, response.getStatus());

}

}

### jUnit test output



# Glossary (A-Z)

|  |  |
| --- | --- |
| **Name** | **Description** |
| Assignment | A task for an operator. |
| Back Office Operator | User of desktop application, his goal is to handle the entire payment order management system. |
| Issue | Render a payment order effective. After issuing a payment order, it must be paid by the debtor. |
| Not pertinent | If the debtor won the legal case, the payment order would be archived. The debtor won’t have to pay the payment order anymore. |
| Payment order | Document which requires the debtor to pay within a defined deadline. |
| Protocol number | Every issued payment order is labelled by an index, a protocol number, which is considered a unique identifier. |
| Reading | Reading of water consumption reported on the meter. |
| Readings Operator | An operator whose task is to perform the reading of water consumption from a meter. |
| Suspend | A payment order can be suspended in presence of a legal case. |