Utility Payments

Analysis and Design Document

Student: Mocan Ioana

**Group:30233**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <05/04/2017> | <1.0> |  | Mocan Ioana |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

The application is an application which brings together all the bills and count of them is made. This application could be very useful for people that work a lot and can not pay.

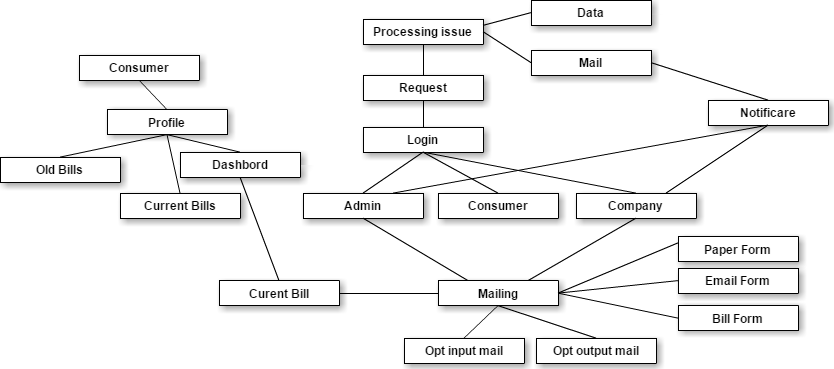
This system will have three different kinds of users:

* Authentication process for the users, administrator and for the utility companies.
* Two different types of registration by using vital information to difference them.
* The users will have at first to choose form a list all the utility companies that they use.
* The users will next have to complete personal information about them so they can connect with the companies.
* The users will receive confirmation that they been inserted in a database and the companies will send the payments here.
* The companies will have to complete information about them and a bank account in which the people will send money for payment
* The companies will have to send bills every month
* The administrator will take care of the site will maintain it in good condition
* The user can make different changes to the profile and update data about him
* The companies can also make different changes about them

# Elaboration – Iteration 1.1

# Domain Model

Domain Modeling is a way to describe and model real world entities and the relationships between them, which collectively describe the problem domain space. Derived from an understanding of system-level requirements, identifying domain entities and their relationship provides an effective basis for understanding and helps practitioners design systems for maintainability, testability and incremental development.



# Architectural Design

## Conceptual Architecture

Model–View–Controller (MVC) is a [software architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) for implementing [user interfaces](https://en.wikipedia.org/wiki/User_interface) on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development. I want to use this pattern because is one of the three ASP.NET frameworks for creating web application and it is very easy to use and can be very useful for understanding the separation between the layers . ASP.NET MVC targets developers who are interested in patterns and principles like [test-driven development](http://en.wikipedia.org/wiki/Test-driven_development), [separation of concerns](http://en.wikipedia.org/wiki/Separation_of_concerns), [inversion of control](http://en.wikipedia.org/wiki/Inversion_of_control) and [dependency injection](http://en.wikipedia.org/wiki/Dependency_injection) . This framework encourages separating the business logic layer of a web application from its presentation layer.

Client Browser

C

View

Post/Get

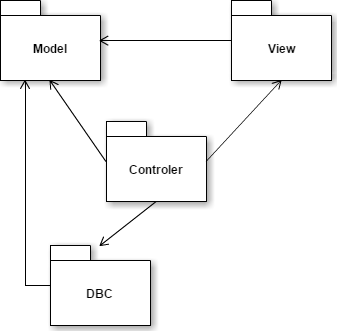
Model

Response

DB

Server

## Package Design



## Component and Deployment Diagrams

Web Server

Utilities

DB Server

SQL Server

User

Any browser

HTML

http

Laptop

Online payment

Persistence

Component Customer

Component Bills

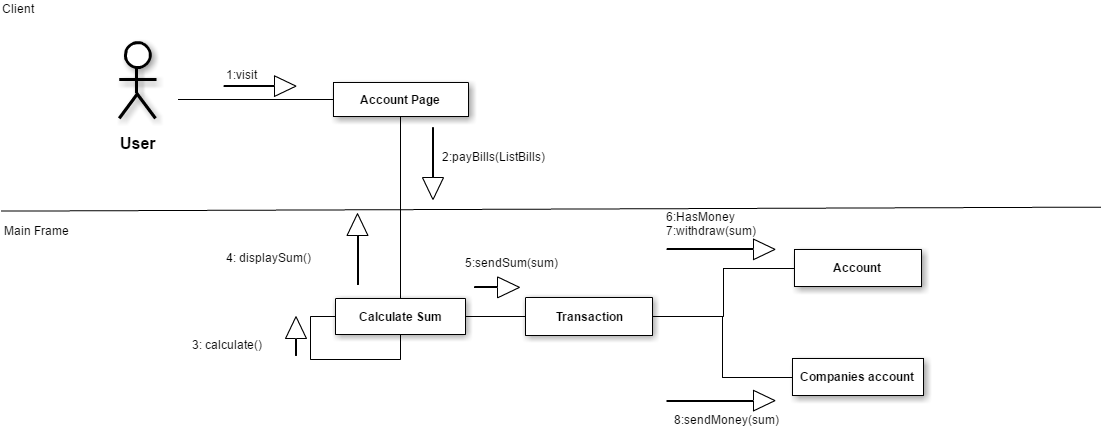
Security

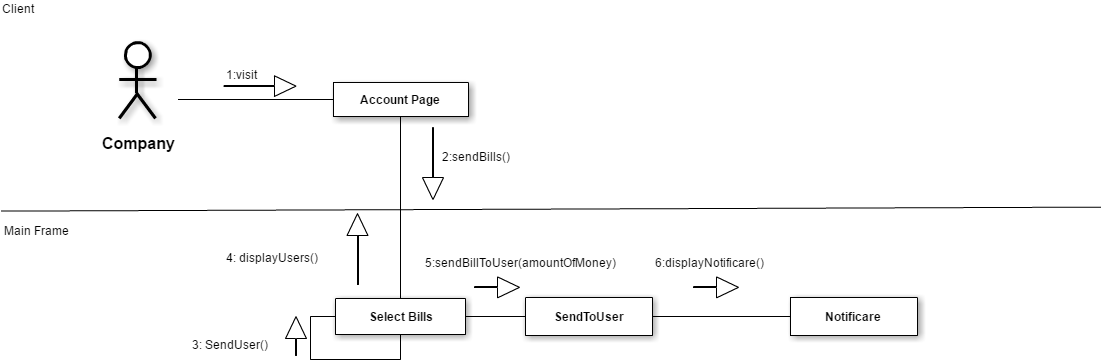
encritare

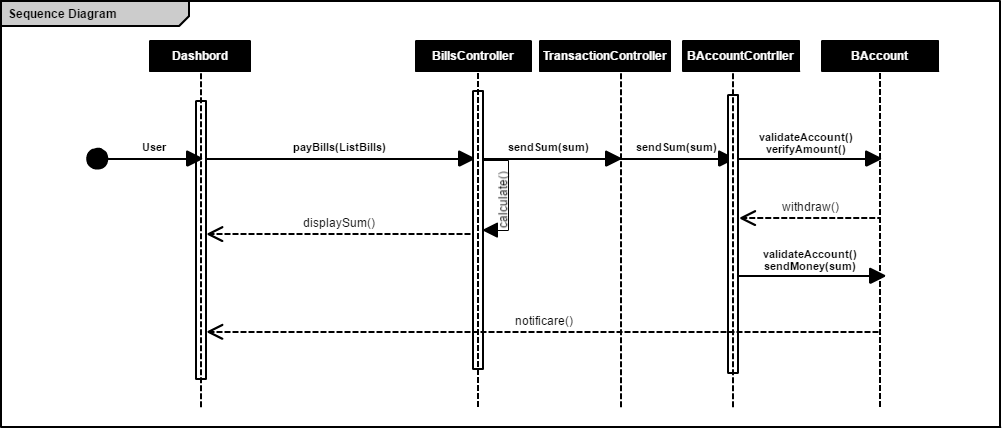
# Elaboration – Iteration 1.2

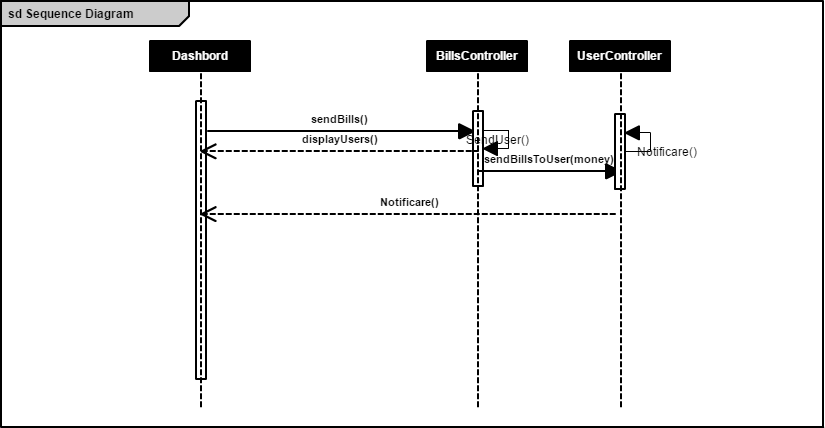
# Design Model

## Dynamic Behavior





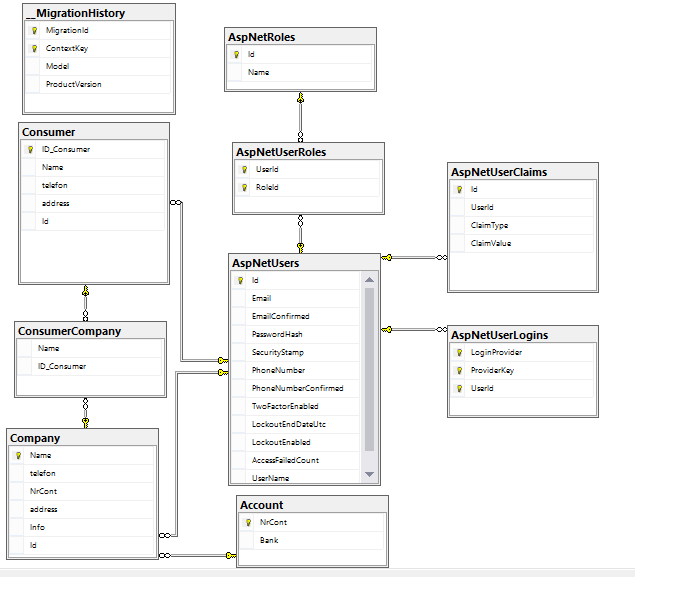




## Class Design

Two design patterns that I want to use are Façade and Observer. Façade I want to use for the mailing part where I want to make the types of forms and the type of mailing if is output or input which will be subsystems. Façade provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use. Observer I want to use for sending notification from the user when the bill is payed to the company and even to the user it self so he can see if the company received the payment. Observer define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

# Data Model



# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*

# Bibliography