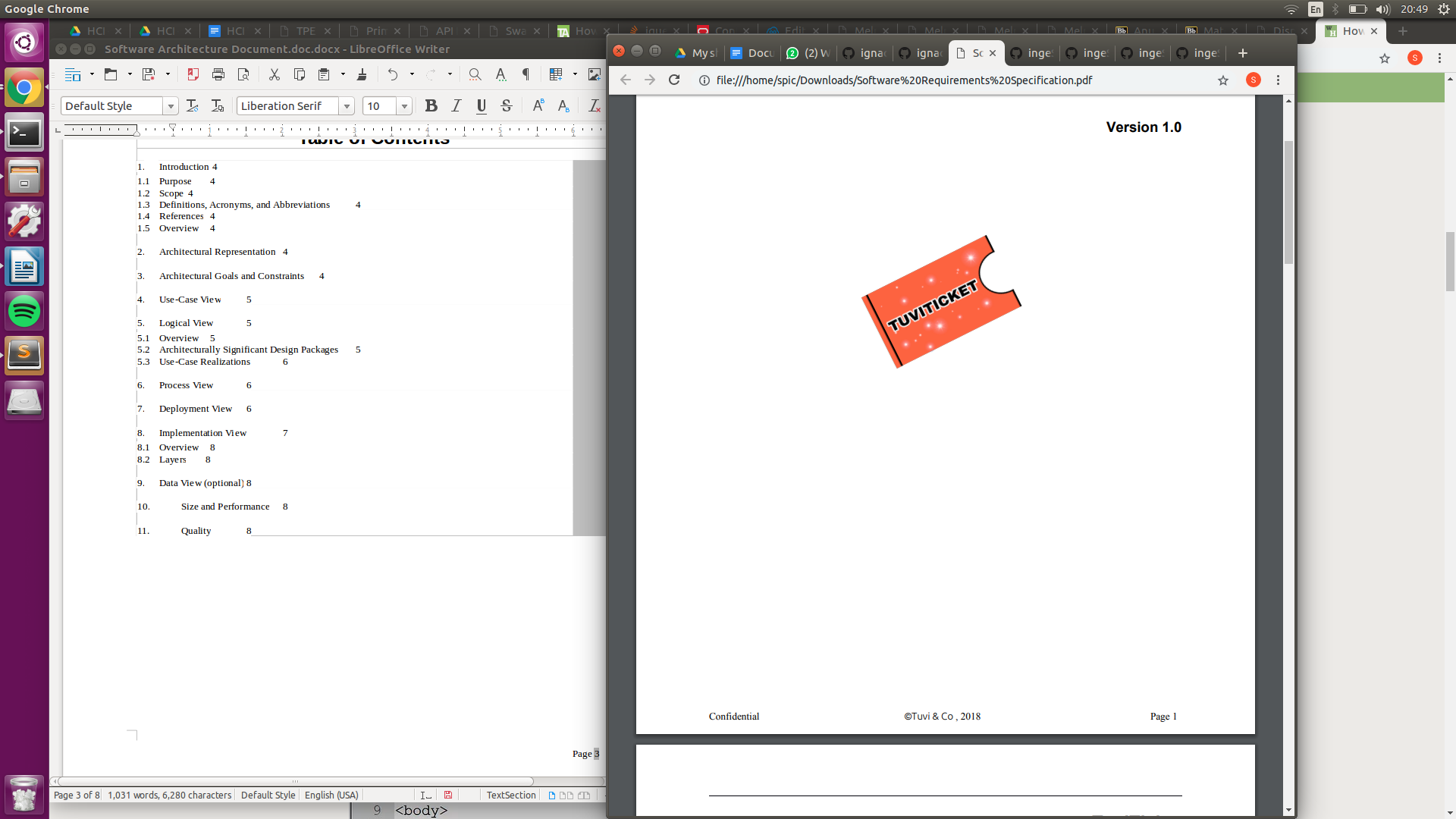
**Software Architecture Document**

**Version <1.0>**



**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 03/05/2018 | 1.0 | First version | Ignacio Vidaurreta  Sofía Picasso  Nicolás Barrera  Juan Bensadon  Nicolas Becker  Ezequiel Keimel |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Architectural Representation 4

3. Architectural Goals and Constraints 4

4. Use-Case View 5

5. Logical View 5

5.1 Overview 5

5.2 Architecturally Significant Design Packages 5

5.3 Use-Case Realizations 6

6. Process View 6

7. Deployment View 6

8. Implementation View 7

8.1 Overview 8

8.2 Layers 8

9. Data View (optional) 8

10. Size and Performance 8

11. Quality 8

**Software Architecture Document**

# **Introduction**

## **Purpose**

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

## **Scope**

TuviTicket is an interface between event organizers and customers. Customers of TuviTicket can find a centralized site to find events near them that might interest them and an easy to use way to buy tickets for said event. They will be able to search for events by category or by most popular and find more information on any event that might interest them. Once they find the event they are looking for they can add it to the shopping cart and proceed with the payment. For each ticket in the cart the customer can choose a payment method to complete the transaction (cash, credit card, etc.) Event organizers can use TuviTicket as a place to promote their events and increase their sales. On creating an event they must include information such as name of the event, number of tickets on sale, pictures, etc.

## **Definitions, Acronyms, and Abbreviations**

See Glossary

## **References**

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirement Specifications. IEEE Computer Society, 1998.

## **Overview**

The following sections provide a more in-depth description regarding the different aspects of the system.

# **Architectural Representation**

The views necessary to describe the software architecture and its representation are:

* Home Screen, which will show the best-selling shows to attract users and access to the other pages, as well as a log-in option
* Search page, which will showcase the results of the user’s search, as well as a search bar for the user to type in
* Cart page, which will contain all the tickets the user has selected to purchase
* Purchase page, which will contain text bars for the user to input the information necessary to complete the purchase, such as credit card number, necessary identification, etc.

# **Architectural Goals and Constraints**

Maximum security is required in all areas concerning monetary transactions. The privacy of users should be guaranteed as they trust their credit information to us.

As for the code, it is expected to have high portability.

# **Use-Case View**

In case user information is required, the system informs if errors were made. When information is displayed to the user, the system shows the information if it exists, or to inform if an error occurred.

Central functionality of the final system in use-cases:

* Use Case 1: User registration
* Use Case 7: Ticket search
* Use Case 10: Adding tickets to the cart
* Use Case 12: Provider registration
* Use Case 13: Product ABM

# **Logical View**

The class diagram is shown in the file Diagrama de Clases.pdf.

The two main actors are the Buyer and the Seller, both Users. Sellers provide tickets to entertainment spectacles, with pictures and information. Buyers can purchase these tickets from other users. There is also the Admin, who interacts with the system to manipulate existing Users.

The model has classes regarding user experience, such as Cart (with all the products marked to be purchased by the user) and Product (with information about the ticket).

There system will also have a view with all the visual elements to be displayed on the page.

Thee controller will handle all interactions between the view and the model. It will also update the database when modifications are made to the model.

## **Overview**

N/A

## **Architecturally Significant Design Packages**

The system will be developed utilizing a MVC oriented development strategy, thus Model, View and Controller will be separated from each other within their own packages. In View we will have the methods that interact with the users, while Model will contain all the information in the system and the Controller will be the middle-man between the two of them.

## **Use-Case Realizations**

All 13 use-cases are available in the project hub, in the file Diagrama de Casos de Uso.

# **Process View**

A web framework will handle communication between the modules and software layers across the solution.

# **Deployment View**

Due to it being a web page, most browsers (such as Google Chrome or Mozilla Firefox) should be able to run the software.

# **Implementation View**

The software will be written in HTML and JavaScript.

## **Overview**

N/A

## **Layers**

# **Data View (optional)**

N/A

# **Size and Performance**

TuviTicket will have a fast and efficient software so that both, buyer and seller, can upload, sell and purchase tickets easily.

# **Quality**

The primary concern in the architecture of the TuviTicket app will be reliability since our customers will provide personal financial information. There will be also a focus on the design to improve the users’ experience.