HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Software Design Document

Version 1.3

Travelwala

Subject: Management of Software Development

Group 10

*Hanoi, January 2023*

Table of Contents

Table of Contents 1

1 Introduction 5

1.1 Objective 5

1.2 Scope 5

1.3 Glossary 5

1.4 References 5

2 Overall Description 7

2.1 General Overview 7

2.2 Assumptions/Constraints/Risks 7

2.2.1 Assumptions 7

2.2.2 Constraints 7

2.2.3 Risks 8

3 System Architecture and Architecture Design 9

3.1 Architectural Patterns 9

3.2 Interaction Diagrams 9

3.3 Analysis Class Diagrams 9

3.4 Unified Analysis Class Diagram 9

3.5 Security Software Architecture 9

4 Detailed Design 10

4.1 User Interface Design 10

4.1.1 Screen Configuration Standardization 10

4.1.2 Screen Transition Diagrams 10

4.1.3 Screen Specifications 10

4.2 Data Modeling 10

4.2.1 Conceptual Data Modeling 10

4.2.2 Database Design 10

4.3 Non-Database Management System Files 11

4.4 Class Design 11

4.4.1 General Class Diagram 11

4.4.2 Class Diagrams 11

4.4.3 Class Design 11

5 Design Considerations 13

5.1 Goals and Guidelines 13

5.2 Architectural Strategies 13

5.3 Coupling and Cohesion 14

5.4 Design Principles 14

5.5 Design Patterns 14

**List of Figures**

# Introduction

This is software design document for Capstone Project about a system that allows user to rent and return bike automatically.

## Objective

This SDD is written for the purpose of giving the audience a clear view about the design of the software. The document’s intended audience is stakeholder and software designer/developer.

## Scope

<*In this subsection:*

1. *Identify the software product(s) to be produced by name*
2. *Explain what the software product(s) will, and, if necessary, will not do*
3. *Describe the application of the software being specified, including relevant benefits, objectives, and goals*
4. *Be consistent with similar statements in higher-level specifications if they exist*

*This should be an executive-level summary. Do not enumerate the whole requirements list here*

*Note that this will be similar to what was written in the SRS.*

>

## Glossary

*<Listing and explaining the terms appearing in the software’s profession and this document. Any assumption of the reader’s prior knowledge or experience on the subject is ill advised>*

## References

|  |  |
| --- | --- |
| [1] | Centers for Medicare & Medicaid Services, "System Design Document Template," [Online]. Available: https://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SystemDesignDocument.docx. |

# Overall Description

## General Overview

About the system, we have some characteristics that make the apps resemble e-commercial website/software: An interface for interacting with user; user request by clicking on the interface and then the request is processed by system controller; we have a database (remote) to store any kind of data; any data-related request or change will be queried in the database; the change in database is then reflected in the UI (user interface). As you can see, there are three main components in the system: the UI, controller and data model. We choose this software to be a desktop application. We choose the three-layer architecture to be our design approach. The design architecture helps to separate different components and better organize the codebase.

Here is general use-case diagram to help you understand the core of our design:

*Fig 1. General Use Case Diagram*

## Assumptions/Constraints/Risks

### Assumptions

User who uses the software should have a good connection to the Internet. Also, our software is a desktop application, so the user also must have a laptop/desktop with an OS (we recommended 64 Bit Microsoft Windows 8 or later; macOS 10.13 or later; or any Linux distribution that supports running application) to run the apps. About the system requirement, we would say 2 GB RAM minimum, 8 GB RAM recommended; for storage 2.5 GB and another 1 GB for caches minimum, solid-state drive with at least 5 GB of free space recommended; require latest version of JRE; 1024×768 minimum screen resolution, 1920×1080 is a recommended screen resolution.

### Constraints

*<Describe any global limitations or constraints that have a significant impact on the design of the system’s hardware, software and/or communications, and describe the associated impact. Such constraints may be imposed by any of the following (the list is not exhaustive):*

* *Hardware or software environment*
* *End-user environment*
* *Availability or volatility of resources*
* *Standards compliance*
* *Interoperability requirements*
* *Interface/protocol requirements*
* *Licensing requirements*
* *Data repository and distribution requirements*
* *Security requirements (or other such regulations)*
* *Memory or other capacity limitations*
* *Performance requirements*
* *Network communications*
* *Verification and validation requirements (testing)*
* *Other means of addressing quality goals*
* *Other requirements described in the Requirements Document*

*>*

### Risks

*<Describe any risks associated with the system design and proposed mitigation strategies.>*

# System Architecture and Architecture Design

Architecture Design steps:

1. Find out software components -> use cases
2. Find out Interaction between use cases
3. Find out Relationship between use cases
4. Draw UML Diagram includes: interaction diagram and analysis class diagram

## Architectural Patterns

In our project, we use Model-View-Controller architectural pattern. There are many benefits of separating classes into Model, View and Controller. And the most important thing is it allows us to update a specific part of an application independently of the other parts

## Interaction Diagrams

## Analysis Class Diagrams

## Unified Analysis Class Diagram

## Security Software Architecture

# Detailed Design

## User Interface Design

### Screen Configuration Standardization

***Display***

Number of colors supported: 16,777,216 colors Resolution: 1366 × 768 𝑝𝑖𝑥𝑒𝑙𝑠

***Screen***

Location of standard buttons: At the bottom (vertically) and in the middle (horizontally) of the frame

Location of the messages: Starting from the top vertically and in the middle horizontally of the frame down to the bottom.

Display of the screen title: The title is located at the top of the frame in the middle.

Consistency in expression of alphanumeric numbers: comma for separator of thousand while strings only consist of characters, digits, commas, dots, spaces, underscores, and hyphen symbol.

***Control***

Size of the text: medium size (mostly 24px). Font: Segoe UI. Color: #000000

Input check process: Should check if it is empty or not. Next, check if the input is in the correct format or not

Sequence of moving the focus: There will be no stack frames. Each screen will be separated. However, the manual is considered a popup message, as the main screen cannot be operated while the manual screen is shown. After the opening screen, the app will start with splash screen, and then the first screen (home screen) will appear.

***Direct input from the keyboard***

There will be no shortcuts. There are back buttons to move back to the previous screen. Also, there is the close button “X” located at the title bar to the right to close the screen.

***Error***

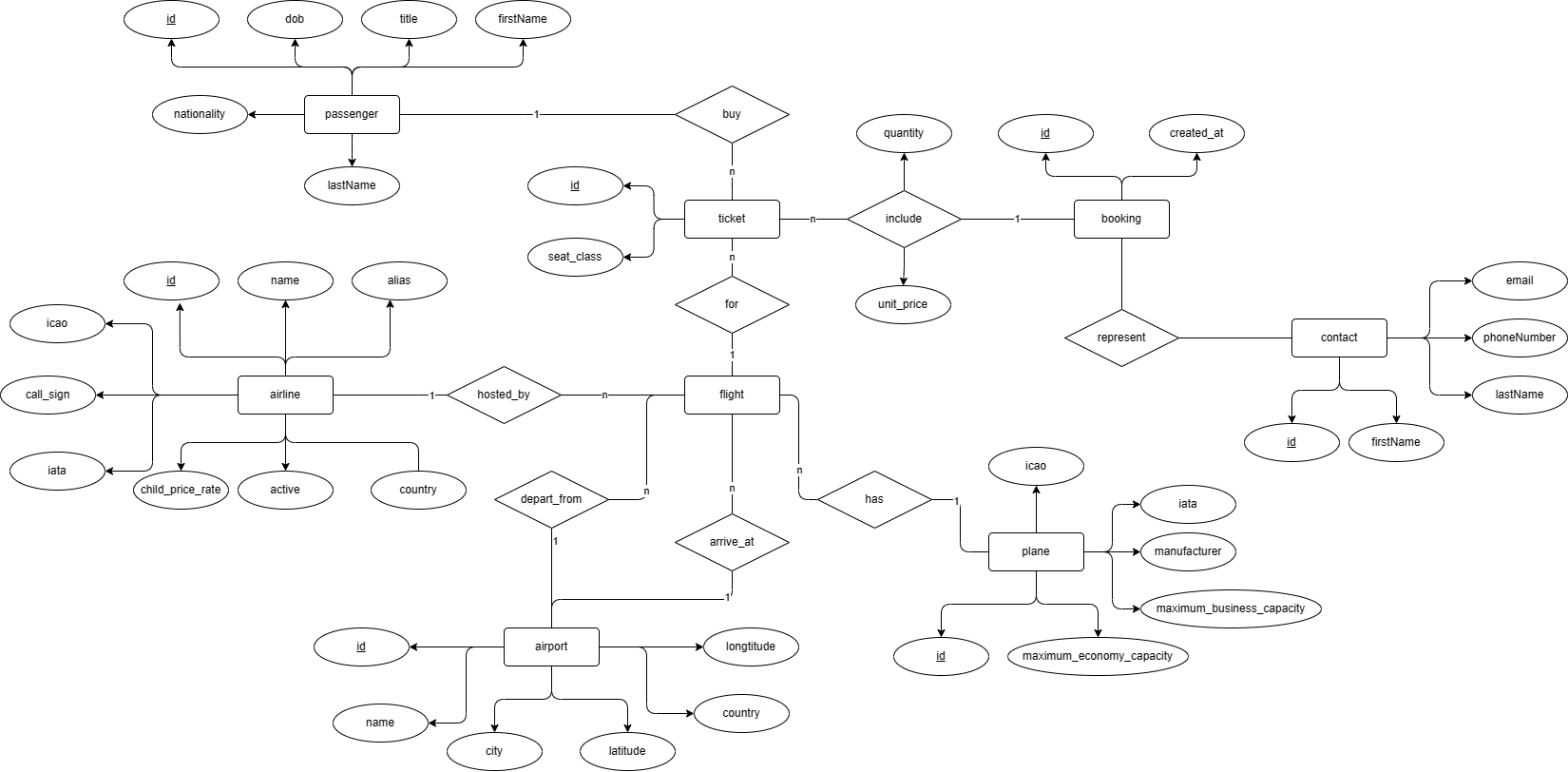
A message will be given to notify the users what is the problem.

### Screen Transition Diagrams

### Screen Specifications

## Data Modeling

### Conceptual Data Modeling



### Database Design

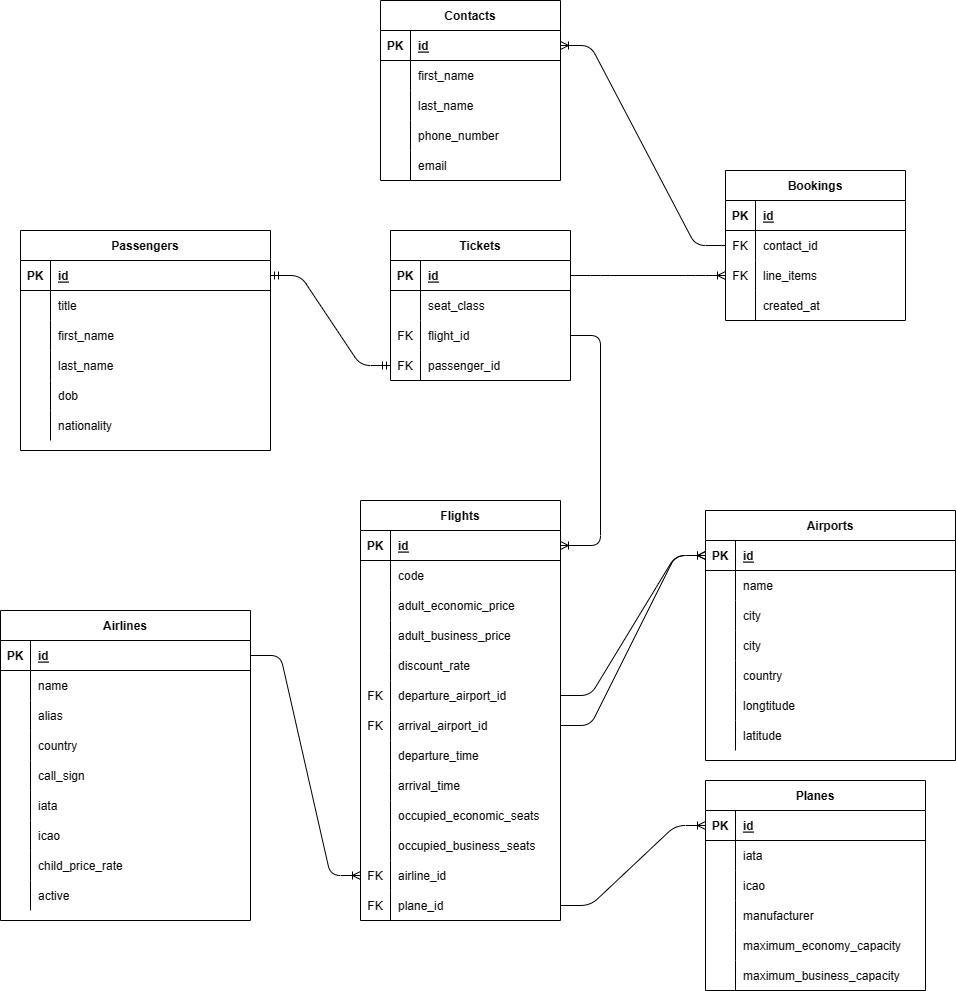
#### Database Management System

MongoDB is a NoSQL database management system that has higher speed and higher  
performance than relational database management system (RDBMS). There is no data  
constraint in MongoDB, hence we do not need to spend time on checking data constraints as we do in RDBMS.

Most SQL databases require scaling up vertically (migrate to a larger, more expensive  
server) when exceeding the capacity requirements of your current server. Conversely,  
most NoSQL databases allow us to scale-out horizontally, meaning we can add cheaper,  
commodity servers whenever we need to.

Queries in NoSQL databases can be faster than SQL databases. Data in SQL databases is  
typically normalized, so queries for a single object or entity require you to join data from  
multiple tables. As tables grow, the joins can become expensive. However, data  
in NoSQL databases is typically stored in a way that is optimized for queries. The rule of  
thumb when using MongoDB is data that is accessed together should be stored together.  
Queries typically do not require joins, so the queries are very fast.

#### Database Diagram



#### Database Detail Design

* **Table Design**
* **flights**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the flight |
| 2 |  | adult\_economic\_price | Double | Price for an adult economic seat |
| 3 |  | adult\_business\_price | Double | Price for an adult business seat |
| 4 |  | discount\_rate | Double | Discount rate |
| 5 |  | departure\_airport | Object | Departure airport |
| 6 |  | arrival\_airport | Object | Arrival airport |
| 7 |  | departure\_time | Timestamp | Departure time |
| 8 |  | arrival\_time | Timestamp | Arrival time |
| 9 |  | occupied\_economic\_seats | Double | Currently occupied economic seats |
| 10 |  | occupied\_business\_seats | Double | Currently occupied business seats |
| 11 |  | airline | Object | Airline that takes care of the flight |
| 12 |  | plane | Object | Plane that is used for the flight |

* **airports**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the airport |
| 2 |  | name | String | Name of the airport |
| 3 |  | city | String | City in which the airport is located |
| 4 |  | country | String | Country in which the airport is located |
| 5 |  | longitude | Double | Longitude of the airport |
| 6 |  | latitude | Double | Latitude of the airport |

* **airlines**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the airline |
| 2 |  | alias | String | Short name of the airline |
| 3 |  | country | String | Country that the airline belongs to |
| 4 |  | call\_sign | String | International call sign of the airline |
| 5 |  | iata | String | International IATA code of the airline |
| 6 |  | icao | String | International ICAO code of the airline |
| 7 |  | child\_price\_rate | Double | Price rate for a child according to adult price |
| 8 |  | active | Boolean | Whether the airline is active |

* **planes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the plane |
| 2 |  | iata | String | International IATA code of the plane |
| 3 |  | icao | String | International ICAO code of the plane |
| 4 |  | manufacturer | String | Manufacturer of the plane |
| 5 |  | maximum\_economy\_capacity | Integer | Maximum seats for economy class |
| 6 |  | maximum\_business\_capacity | Integer | Maximum seats for business class |

* **passengers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the passenger |
| 2 |  | first\_name | String | First name of the passenger |
| 3 |  | last\_name | String | Last name of the passenger |
| 4 |  | dob | Timestamp | Date of birth of the passenger |
| 5 |  | nationality | String | Nationality of the passenger |
| 6 |  | title | String | Title of the passenger: Mr, Ms |

* **contacts**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the contact |
| 2 |  | first\_name | String | First name of the contact |
| 3 |  | last\_name | String | Last name of the contact |
| 4 |  | phone\_number | String | Phone number of the contact |
| 5 |  | email | String | Email of the contact |

* **tickets**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the ticket |
| 2 |  | seat\_class | String | Seat class of the order (economic, business) |
| 3 |  | flight\_id | ObjectId | ID of the flight for the ticket |
| 4 |  | passenger\_id | ObjectId | ID of the passenger that books this ticket |

* **bookings**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *#* | *PK* | *Column name* | *Data type* | *Description* |
| 1 | x | id | ObjectId | ID of the booking |
| 2 |  | contact\_id | ObjectId | The representative of the booking |
| 3 |  | line\_items | Array | Array of tickets for the booking |
| 4 |  | created\_at | Timestamp | Time when the booking is processed |

## Non-Database Management System Files

## Class Design

### Class Diagrams

### Class Design

# Design Considerations

## Goals and Guidelines

**Goals:**

* Bring a good looking and good experience for users
* The response time for the system is 1 second at normal (do not let user have to wait)
* All errors when using application will be informed to user immediately

**Guidelines:**

* Observe java convention in coding, OOP principles
* Explain code, write java doc for maintenance

## Architectural Strategies

Our design decisions focus on reusing components, unified system following

+ Programing Language: Java

+ Database: MongoDB

+ Unified on error detection and recovery

We always toward save memory and spaces, also speed up response time and nice looking. In the future, we plan to extend software:

* Have site for admin to add, delete flights
* Statistics
* Business strategies
* Realtime system with multi-thread

These targets make us concentrate totally on architectural design.

## Coupling and Cohesion

### Coupling

### Cohesion

## Design Principles

### S – Single Responsibility Principle:

|  |  |  |  |
| --- | --- | --- | --- |
| Related module | Description | Improvement | Done |
|  |  |  |  |

### O – Open-Closed Principle:

|  |  |  |  |
| --- | --- | --- | --- |
| Related module | Description | Improvement | Done |
|  |  |  |  |

### L – Liskov Substitution Principle:

|  |  |  |  |
| --- | --- | --- | --- |
| Related module | Description | Improvement | Done |
|  |  |  |  |

### I – Interface Segregation Principle:

### D – Dependency Inversion Principle:

|  |  |  |  |
| --- | --- | --- | --- |
| Related module | Description | Improvement | Done |
|  |  |  |  |

## Design Patterns

### Singleton Pattern:

Singleton pattern ensures that a class has just a single instance and provide a global access point to that instance.

|  |  |  |  |
| --- | --- | --- | --- |
| Related module | Description | Improvement | Done |
|  |  |  |  |

### Strategy pattern:

|  |  |  |  |
| --- | --- | --- | --- |
| Related module | Description | Improvement | Done |
|  |  |  |  |