MyID90 Final Report

Carson Holscher, Chis Kim, Thomas Bioren

5/15/2024

Contents

[Executive Summary 3](#_Toc166759269)

[Introduction 3](#_Toc166759270)

[Problem Description 3](#_Toc166759271)

[Solution 4](#_Toc166759272)

[Front-End 4](#_Toc166759273)

[Back-End 4](#_Toc166759274)

[Key Challenges 5](#_Toc166759275)

[Database Design 5](#_Toc166759276)

[Security Measures 5](#_Toc166759277)

[Integrity Constraints 6](#_Toc166759278)

[Stored Procedures 6](#_Toc166759279)

[Views 7](#_Toc166759280)

[Indices 7](#_Toc166759281)

[Triggers 7](#_Toc166759282)

[Design Analysis 8](#_Toc166759283)

[Strengths 8](#_Toc166759284)

[Weaknesses 8](#_Toc166759285)

# Executive Summary

This final report introduces the reader with the problem that serves as the basis for MyID90. Then, it delves into the solution we implemented, followed by major issues we overcame. We follow this with the overall design on our database and frontend application. Finally, we discuss the strengths and weaknesses of our solution to the problem. Included at the end is an appendix and external references for the reader’s convenience.

# Introduction

This is MyID90’s final report, written by Thomas Bioren, Carson Holscher, and Chris Kim. This paper’s purpose is to address MyID90’s solution to the given problem and its advantages and disadvantages. Additionally, we will readdress topics from the Security Analysis and Final Problem Statement.

# Problem Description

To fully understand our problem, some background knowledge is provided. Airlines provide tickets to airline industry members at a 90% industry discount (ID90). However, an employee is not guaranteed a seat on the airplane when purchasing this ticket, they have only purchased a *standby* ticket. Purchasing a standby ticket—or flying standby—means the purchaser is only permitted on the airplane if there are open seats.

The inherit problem arises because airlines treat how full a flight is (its load) as secret information only accessible by employees of that specific airline. As an example, this means that an employee at Alaska Airlines could purchase an ID90 ticket on Lufthansa but would not know if they will get a seat on that flight. To put it plainly, the problem MyID90 is trying to address is the inability of airline industry employees to accurately predict if they will get a seat on a flight they booked with an ID90 ticket.

A current solution to this problem is a popular Facebook group for airline employees to ask other employees to look up flight loads for them via their employee portals. To build on the example from the previous paragraph, the Alaska Airlines employee could post on the Facebook group asking a Lufthansa employee to look up the load for the Alaska employee’s specific flight and report it back. MyID90 aimed to replace this Facebook group. The goal of this project was to create an application that users could log into and request and fulfill load requests.

In the table below are the features in MyID90’s final problem statement.

|  |  |  |
| --- | --- | --- |
| Feature Number | Feature Name | Feature Description |
| 1 | Submit Load Request | Users can request another user look up the load on a specific flight. |
| 2 | Fulfill Load Request | Users can fulfill another user’s load request to earn that request’s token value. |
| 3 | Trip Planner | Users can “save” flights for future reference in a list. |

# Solution

MyID90 settled on a Java application using Java Swing for its graphics. The database backend is run by Microsoft SQL Server.

## Frontend

The frontend of the MyID90 application was written in Java with Java Swing and FlatLaf to improve its appearance. To ensure usability, we separated the application into different windows that can be navigated to from the home screen. Each screen takes a minimalistic design perspective and only displays information that is important to the user. An important note is that there are two classes of user: Employee and Dependent. Dependents can do everything an Employee can apart from creating and fulfilling load requests and managing their dependents (as they do not have any).

Employees can create load requests in the Manage Load Requests screen. In this screen, the user sees their active load requests and their most recent load. The user can also submit new load requests here.

Employees can fulfill load requests submitted for flights operated by their airline on the Fulfill Load Requests screen. There, the user can select a load request and enter its new load.

Employees can manage their dependents in the Manage Dependents screen. There, they can receive their add code, remove unwanted dependents, and re-add deleted ones.

Additionally, there are extra tabs to add new flights, update their info, delete destinations, and view flight information.

## Backend

The GUI is connected to the MyID90 database. This database contains tables with information about employees and their dependents, flights, airlines, destinations, and load requests. The application account used for MyID90 is only given permission to use the stored procedures necessary to run the application and no more. This ensures the database is as secure as practical. The database itself is run on the Rose-Hulman CSSE server and will be removed at the end of the Spring 2024 quarter. More information is available in the database section.

## Key Challenges

* **Challenge:** Java Swing is not a good UI library.

**Solution:** Use Flatlaf, a tool that makes Java Swing a little better looking.

**Analysis:** We did the best we could with Java Swing. As other teams have noted, it is a hard tool to use. Our graphics may not be very pretty, the GUI is usable.

* **Challenge:** Creating a non-abusable method of registering dependents.

**Solution:** We created an add code that a dependent must enter to register as an employee’s dependent.

**Analysis:** This additional step ensures a dependent can only register as an employee’s dependent if they have that employee’s add code. This code is only accessible by the employee. This ensures .

* **Challenge:** Communication throughout this complicated project

**Solution:** All team members were members of a Microsoft Teams team. This team is where we would communicate and plan our next milestones.

**Analysis:** This solution was unreliable. Team members did not regularly check Teams and were unable to be contacted for large swaths of the day. This caused much unnecessary stress and many miscommunications throughout the lifespan of this project.

# Database Design

## Security Measures

The application uses a special user account that can only interact with the database through specific stored procedures. That means the application’s functions for interacting with the database must use those stored procedures. This prevents SQL injection attacks by making all inputs be treated as parameters.

We also take precautions against the application account being compromised by encrypting its password, making it much more difficult to access outside of the application.

## Integrity Constraints

Here is a list of integrity constraints used in our database:

* Employees & dependents may not have null first & last names

## Stored Procedures (INCOMPLETE)

|  |  |
| --- | --- |
| Stored Procedure Name | Stored Procedure Purpose |
| CreateAirline | Inserts an airline into the database, if no airline with its ID already exists |
| CreateDependent | Adds a dependent & a user to the database, if its employee exists & its username isn’t taken. |
| CreateDestination | Adds a destination to the database. If its IATACode already exists, update it instead |
| CreateEmployee | Adds a new user & employee to the database, if the username isn’t taken |
| CreateFlight | Adds a flight to the database |
| GetUser | Gets a user’s password hash & salt |
| CreateLoadRequest | Creates a load request, if one doesn’t exist for that employee & flight combination |
| CreateUpdate | Adds an entry to the updates table, if one with that ID doesn’t exist. |
| CreateUserDependent | Adds a dependent & a user to the database, if its employee exists & its username isn’t taken. |
| CreateUserEmployee | Adds a new user & employee to the database, if the username isn’t taken |
| DeleteLoadRequest | Sets a load request to be invisible, returns its submitter’s tokens, & shows the IsVisible column for the given load request |
| DropAirline | Sets an airline to be invisible |
| DropDestination | Sets a destination to be invisible |
| DropDependent | Removes a dependent from the database |
| DropFlight | Sets a flight & any LoadRequests & Updates to be invisible |
| GetDependents | Lists all dependents for a given employee |
| GetEmployeeLoadRequests | Lists all load requests for a given employee |
| GetFlight | Gets the information on one specific flight |
| GetTokens | Returns the number of tokens that the user has |
| isDependent | Returns information if the user is a dependent |
| ListAirlines | Lists the ID column of every visible airline |
| ListDestinations | Lists every the IATACode & name of every visible destination |
| ListsFlightsConditions | Returns info on flights based on a variety of conditions. Used for search bars |
| ListsFlights | Lists all flights from the user’s airline |
| ListLoadRequests | Lists all of a user’s load requests |
| ListWholeFlight | Lists all information about a flight. Used for the View Flight Page |
| RemoveDependent | Removes a dependent from an employee’s dependent list |
| UnRemoveDependent | Returns a removed dependent to an employee’s dependent list |
| UpdateFlight | Updates a flight’s information |
| UpdateLoadRequestCost | Changes the token cost of a given load request |
| UpdateLoadRequest | Fulfills a load request, resulting in a flight having its load updated & the user gaining 1 token. It also adds an appropriate entry to the updates table |

# Views

The MyID90 database does not contain any views.

# Indices

# Triggers

The MyID90 database does not contain any triggers

# Design Analysis

## Strengths

## Weaknesses