MyID90 Final Report

Carson Holscher, Chistopher Kim, Thomas Bioren

5/17/2024

Contents

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 inherent 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.

Additionally, if an attacker were to get direct access to the database through the application account, the account is limited to only executing specific stored procedures. This renders the attack relatively useless.

Integrity Constraints

Commented [BT1]: Finish this!!!

Commented [TB2]: @Bioren, Thomas I want to look at this.

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

- Employees and dependents may not have null first or last names.
- If a database entry that any foreign key depends on would be deleted, reject the deletion.
- If a database entry that is depended on would have its primary key updated, reject the update.

Stored Procedures (INCOMPLETE)

Stored Procedure Name	Description
CreateAirline	Inserts an airline in the database if no airline with its ID already exists.
CreateDestination	Adds a destination to the database. If a destination with the given IATA code already exists, it updates it instead.
CreateFlight	Adds a flight to the database.
CreateLoadRequest	Creates a load request if one does not already exist for the given user and flight.
CreatePlan	Creates a plan, or adds a flight to it, or updates the order of a flight in the plan
CreateUpdate	Adds an entry to the updates table if one with that ID does not already exist.
CreateUserDependent	Adds a dependent and sets their employee if the username is not already taken and

Commented [TB3]: @Holscher, Carson write what this does.

	the provided employee username and add code are correct.
CreateUserEmployee	Creates a new employee user if the username is not already taken.
DeleteLoadRequest	Sets a load request's visible attribute to be false and returns its submitter's tokens.
DropAirline	Sets an airline's IsVisible attribute to false.
DropDependent	Removes a dependent from the database.
DropDestination	Sets a destination's visibility to false.
DropEmployee	Removes an employee from the database.
DropFlight	Sets a flight and any load requests associated with it to not be visible.
DropPlan	Sets a plan's visibility to false
DropPlanItem	Removes a flight from a plan.
GetDependents	Gets all the dependents that have the provided employee as their employee.
GetFlight	Returns the given flight.
getID	Returns the ID associated with a user's username.
GetTokens	Returns the number of tokens a user has.
GetUser	Returns a user's salt and password hash.
isDependent	Returns if a user is a dependent.
ListAirlines	Lists the ID of every visible airline.
ListAllPlans	Lists the name of every visible plan.
ListDestinations	Lists the IATA code and name of every visible destination.
ListFlights	Lists all flights offered by the given user's airline.
ListFlightsConditions	Returns all flights that match the given conditions. Conditions can be null. If condition(s) are null, does not search for flights with that condition parameter.
ListLoadRequests	Returns all the load requests a user has submitted.
ListPlan	Returns all flights on a given plan.
ListWholeFlight	Return all information about a flight. Used for displaying the flight.
listYourLoadRequests	Lists the load requests for a specific user.
RemoveDependent	Sets a dependent's visibility to false. This allows them to be re-added in the future.
UnRemoveDependent	Sets a dependent's visibility to true. This effectively re-adds them.
UpdateFlight	Updates a flights attributes.

Commented [TB4]: Verify this

Commented [TB5]: @Holscher, Carson Verify

Commented [TB6]: @Holscher, Carson Verify

Commented [TB7]: @Holscher, Carson verify

Commented [TB8]: @Kim, Chris Verify

Commented [TB9]: @Holscher, Carson Verify

Commented [TB10]: @Holscher, Carson verify

UpdateLoadRequest	Creates an entry into the Updates table, updates the load of a flight, & increases the user's token count by 1
UpdateLoadRequestCost	Updates a load request's token value.

Commented [TB11]: @Kim, Chris Verify

Views

The MyID90 database does not contain any views.

Indices

The MyID90 database has six indices:

Table	Column indexed	Reason
Flight	DepartureDateTime	It will allow us to easily select flights that occur in the future, which will be important when the database gets older & most flights in it already happened.
SubmitsLoadRequest	DateAndTime	It allows us to prioritize load requests for flights that will happen soonest
Airline	Alliance	It allows us to group alliances together.
Flight	Load	Lets us sort by what flight has the most seats available
Flight	LastLoadUpdate	Lets us know how outdated the load on a flight might be

Triggers

The MyID90 database does not contain any triggers.

Design Analysis

Commented [TB12]: @Bioren, Thomas I want to go over this

Strengths

- The application user can only interact with the database by executing specific stored procedures, meaning the database is protected from SQL injection.
- Stored procedures try to be flexible. When it's feasible, stored procedures that insert into a table will be able to update an entry if it already exists.
- The database uses indices that speed up sorting.
- Table and stored procedure names are descriptive.

Weaknesses

- Some UI features, such as the order columns button, overcomplicate the user interface.
- The password is stored in plain text in the source code. Given more time, we would have encrypted this.

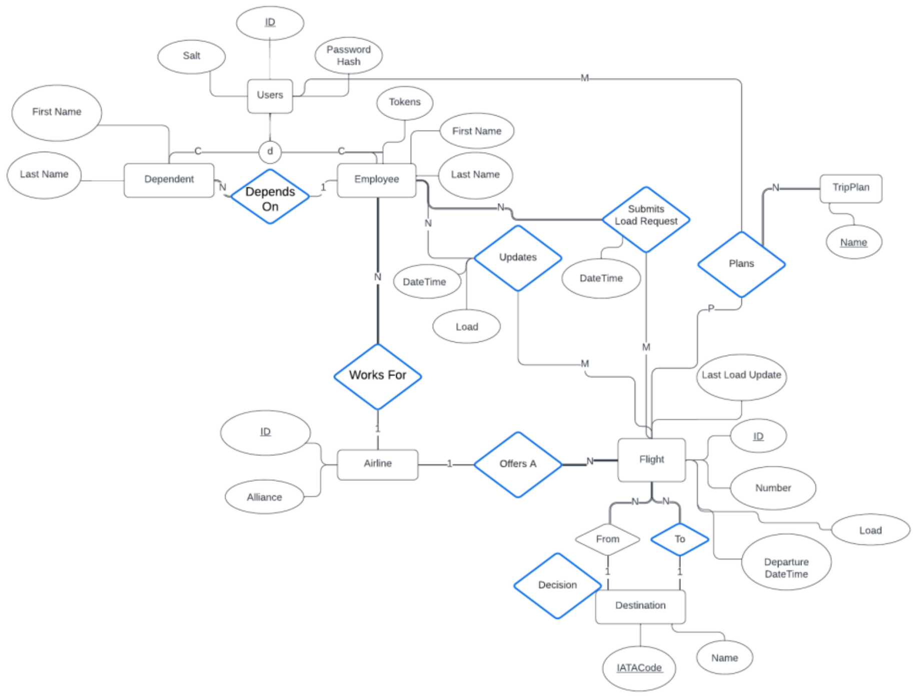
Appendix A

Relational Schema

Relations:
Employee: (ID, FirstName, LastName, Tokens, AirlineID)
Dependent: (ID, FirstName, LastName, EmployeeID)
Users: (ID, Username, Salt, PasswordHash)
Submits Load Request: (EmployeeID, FlightID, DateTime, Token)
Updates: (EmployeeID, FlightID, DateTime, Load, Token)
Flight: (ID, LastLoadUpdate, Number, Load, DepartureDateTime, FromCode, ToCode, AirlineID)
Airline: (ID, Alliance)
Destination: (IATACode)
TripPlan: (UserID, FlightID, PlanName, Order)

Foreign Keys:
Employee(ID) -> Users(ID)
Dependent(ID) -> Users(ID)
Employee(AirlineID) -> Airline(ID)
Dependent(EmployeeID) -> Employee(ID)
Submits Load Request(EmployeeID) -> Employee(ID)
Submits Load Request(FlightID) -> Flight(ID)
Updates(EmployeeID) -> Employee(ID)
Updates(FlightID) -> Flight(ID)
Flight(FromCode) -> Destination(IATA Code)
Flight(ToCode) -> Destination(IATA Code)
Flight(AirlineID) -> Airline(ID)
TripPlan(UserID) -> Users(ID)
TripPlan(FlightID) -> Flight(ID)

ER Diagram



Explanation of ER Diagram

MyID90 has employees and dependents. Each dependent belongs to an employee. This is shown by the “Depends On” relationship. Both employees and dependents have attributes in common that make them “users.” This is demonstrated by the is-a relationship between both employees and dependents and the user entity. Employees work for an airline that has an ID and alliance. Airlines offer flights that have many attributes; the most important of which is “load.” The load attribute is updated by load requests. An employee submits a load request through the “Submits Load Request” relationship. Other employees can update flight loads through the “Updates” relationship. Both these relationships have token values that tell the database to increase or decrease an employee's token count. Finally, users can create plans for flights. This is demonstrated by the three-way relationship named “Plans” that connects “Users,” “Flight,” and “TripPlan.”

Glossary

UI (User Interface): The part of the program that the user directly interacts with.

IATACode: The unique identifier used for airports around the world.

Flying Standby: Purchasing an airline ticket that only guarantees a seat if there are extra seats remaining

ID90: A 90% industry discount for all airline employees who choose to fly standby.

Load: The number of seats available for a flight.

Stored Procedure: SQL code designed to perform a specific task that is stored in the database.

SQL: Structured Query Language. A programming language used for managing databases.

Frontend: The parts of a program responsible for the UI.

Backend: The parts of a program responsible for interacting with the database

References

1. MyID90 Documentation. Security and Data Integrity Analysis document. Delivered on: 4/20/24
2. MyID90 Documentation. Final Project Problem Statement document. Delivered on: 5/10/24
3. MyID90 Documentation. Indices document. Not delivered.

Index