AIRLINE FLIGHT SCHEDULER

SOFTWARE DESIGN DOCUMENT

Team 18
Ryan Witkowski
Nicholas Lawrence
James Fox

INTRODUCTION

PURPOSE

The purpose of the system is to make sure that the database is always able to be accessed. The system should always be available, so uptime is a major factor in the architecture of our system. The system should also be able to handle multiple users, and not be throttled by many users. The system should also provide a user interface that is easy to use, and users should able to complete intended tasks as soon as possible. The system will be used by flight crew and the managers of the flights. The flight crew is not going to need any authentication for the database because they will just need to know the flights that they are on, or someone that they are working with is on. The managers of the flights will need to have authentication because they are the ones that are setting the schedules and changing them as needed.

DESIGN GOALS

The goals of the design are for the system to work consistently and ensure that the design is well thought through. One of the goals is to make sure that the design takes numerous factors into account to prevent errors that could cause the software to fail. The software should also be as efficient as possible, so optimization is critical for the Airline Crew Scheduler software. The system should have a well-designed user interface that is appealing to look at and easy to use.

DESIGN TRADE-OFFS

One of the trade-offs that we are taking into account is the use of Java. We were not sure how our system should be designed; whether it should be oriented for many platforms or just the computer, and that limited us to not being able to access the program from online. This trade-off was made for simplicity reasons and allows for more integration of object-oriented programming. Since the system is going to be around for some time and improvements will be made through maintenance, the system is going to be built for response. This was decided by the fact that most systems that will run this program will have plenty of resources to handle the expected memory use. Another trade-off we faced with our design was the use of the CSE server. This is the location where we will host the database, saving us time with setting up our own server to host our database. The conclusion of this use was decided by using MySQL over SQL, and some of the costs that could have come from using third party services with SQL. Lastly, to tie back in with the use of java, there is high use of memory when running a java application. For instance, C++ would be more efficient for this application because it uses less memory, but the language is more complicated to work with, especially with front end design.

INTERFACE DOCUMENTATION GUIDELINES

- Classes are named with singular nouns.
- Methods are named with verb phrases, fields, and parameters with noun phrases.
- Error status is returned via an exception, not a return value.
- Proper java syntax laid out by Google will be used for the source code.

DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

CSE – Computer Science and Engineering (UNL department)

SQL – Structured Query Language.

OVERVIEW

The system we are designing is a Java program that integrates a MySQL database to schedule and manage flight crews. The system should have no downtime and will be well planned out. The architecture will be able to handle changes of state that may cause an error if interrupted. The Java program will be made so that the two types of users that were stated previously will be able to complete tasks of entering and changing information in the database and looking up information from the database that involves specific flight information. The database will be made so there are backups of the logs in the case of an event failure.

CURRENT SOFTWARE ARCHITECTURE

As the project currently stands, there is no current software architecture fully built outside of the planned design of a three-tier architecture. The tier that is currently in development is the interface, but it is not completely finished at this time. There is a user login screen that helps to filter between authorized and unauthorized personnel, and then presents the appropriate prompts based on the status of the user. This function completes F1 functionality and is the only section of our current software that has been built at this time.

PROPOSED SOFTWARE ARCHITECTURE

OVERVIEW

The Software Architecture that we selected to use for the flight crew scheduler was a 3-Layer Architectural Style. This will allow users to interact with the system in a way that is controllable by the software. We will have a front end to interact with the middleware, which will then be able to make calls to the database. This will allow the system to have security by not allowing users direct access to the database itself.

SUBSYSTEM DECOMPOSITION

The top layer will be the user interface using swing with java. This will be able to handle the input and display the information from the database to the user. To get the data, the interface will have to get information from the middleware that is also in java, and this in turn will interact with the CSE server. Once the system is interacting with the CSE server, it will be able to make calls to the database to either retrieve or send information. This will then provide feedback to the user interface about the transfer of data.

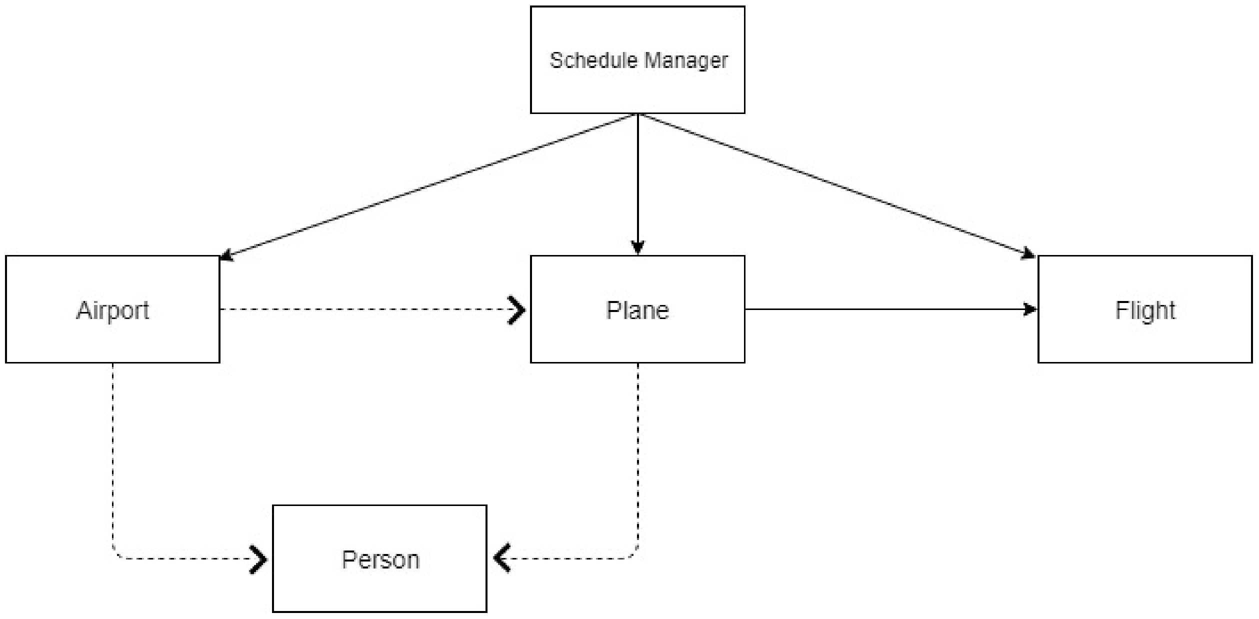


Figure 1 System Decomposition Model

HARDWARE/SOFTWARE MAPPING

This section is not applicable to our design, because we are not directly working with hardware. The system will be able to run on any computer that runs Java. There is no way for us to control any hardware on the CSE server. Lastly, we are limited with hardware control with Java because unlike other languages like C++, Java automatically manages memory use.

PERSISTENT DATA MANAGEMENT

The data will be stored in a MySQL database that is located on the CSE server. The data will be stored in this location and will have the ability to change based on calls from valid users. The database will also have the ability for users to call the data and have it display on the user interface. This will allow the data to always have a way to be accessed at different times. Lastly, the database will have a backup log so that in the event of a database failure, the data will be able to be reloaded into the database that became corrupted once the issue is resolved.

ACCESS CONTROL AND SECURITY

The unauthorized user, for example a member of the flight crew, will not be able to access the data directly within the database. The flight crew will be able to see the data that is in the database, but it will not be able to enter data within the database. The reason for this is so the database will not have unknown users making calls to the database that will disrupt the system. Users that have a valid account, for example an airline manager, will be able to make changes to flight crew schedules, and move around different entities as needed. The airline manager will need to sign into the database to do so.

GLOBAL SOFTWARE CONTROL

The way that the system will be able to define what users have a valid account is by making the valid users log into a separate part of the user interface to be able to enter data within the database. If a user is a valid user, they will be able to log into the system, otherwise they will not be able to sign in. If a user that does not have a login needs to look up flight information, they will be able to do so because the data does not need to be managed as heavily. This will be a control that allows the database to make sure that invalid users are not making changes that are not approved.

BOUNDARY CONDITIONS

When the system has been started on the user's side, there will not be any data that is needed to be displayed immediately. The user will be able to select the location they would like to proceed to, and from there the information will be displayed based on what the user selects. The database will establish the connection to the server upon creation of the database, and when a user starts a session, the connection to the CSE server will be initialized.

When the system is terminated, the connection to the server and database will be terminated as well, and all data that was displayed will be destroyed and collected by the garbage collector within java. The database will not be able to be terminated, and neither will the server. Since the system is working with systems that are controlled by outside resources, termination of the database or server will result in an error message and will return to the home page for the user interface. This will allow the user to know what error has occurred.

As previously stated with the termination, when the system is not able to reach the intended call with the database, these unsuccessful calls will be treated as an error and the system will exit the calls to the database. This will let the user know that the change was not made and will provide information about the error. When the user does not have an internet connection, any call to the database will result in a message to the user saying that their connection to the internet is no longer available and from there they will have the option to check connection again.

SUBSYSTEM SERVICES

The Person subsystem is a lower layer that is utilized by other subsystems in the system. It provides services such as classification and qualifications of certain crew members, and identifies crew members as either captains, first officers, or flight attendants. These services are realized in the form of attributes that belong to each individual type of person. Other subsystems call on the Person subsystem to provide classification information that is necessary in regard to scheduling the crew.

The Flight subsystem is a mid-level layer that provides three services to the Plane and Schedule Manager subsystem. The first of these is the Flight Scheduled service which schedules the appropriate crew to a flight upon its initial creation. The Flight Changed service adjusts the scheduling of crew members in the event that various aspects of a flight are changed from the original creation. The Flight Cancelled service updates the scheduling system in the event of a flight being cancelled altogether. These three services are executed through methods that are called upon when one of these events occurs.

The Plane subsystem is a mid-level layer that depends on the Flight subsystem and calls upon the Person subsystem for functionality. This subsystem provides services that identifies the type of plane to be used in a scheduled flight, along with the personnel that will be on board the plane, and these services are provided via attributes and calls to other subsystems. The information in this subsystem is then utilized by the Airport subsystem.

The Airport subsystem is a mid-level layer that calls upon the Person and Plane subsystems to operate. This subsystem is responsible for identifying which crew members (in-flight or on standby) are located at a certain airport, and the planes that are currently located at a certain airport. These services are realized through characterization attributes and calls to other subsystems and are then utilized by the Schedule Manager subsystem.

The Schedule Manager subsystem is an upper layer that depends on the Airport, Plane, and Flight subsystems to complete the crucial service of scheduling crew members based on the needs and adjustments of flights. Its services are provided by methods that analyze crew availability, fulfill shift hour requirements, and match requirements set by the specific flights that are created. This subsystem depends on almost all layers below it to provide necessary flight and personnel information in order to accurately and efficiently execute its service, and this subsystem is the highest subsystem layer in our system.

PACKAGES

The three packages that we have in our system are the interface, the data manager, and the database. The goal of this three-tier architecture is to have the interface interact with the middleware, which in turn interacts with the database on its behalf and the information is passed back along to be displayed through the interface once again.

In our current system, the interface package is comprised solely of itself, meaning that there are not multiple subsystems as part of the whole package. The package then interacts exclusively with the data manager, and there is no direct interaction with the database. This helps to ensure security in the system and maintain stability through the use of our architecture style.

The data manager package is currently the most complex package of the three and is comprised of the five subsystems outlined under Section 4. These subsystems are responsible for interpreting input from the interface, querying the database according to the information received, and organizing and outputting data back to the interface accordingly. This data manager further increases stability and organization within the system.

The database package is tasked with the main responsibility of receiving, storing, and sending all necessary information within the system, and is a sole subsystem in and of itself. It receives queries from the data manager and returns the appropriate information in a secure and efficient manner. Security of information entered is of the utmost importance, and the architecture employed, along with other precautions taken, help to ensure safe and secure information storage and transferal.

CLASS INTERFACES

The Airport class will take in arguments from the person class. Airport will have an airport name and attribute along with StandbyCrewWorkers attribute, which will pull from persons class. It will push and pull to and from schedule manager, and it will pull from the plane class.

The Captain class will take in one argument, which will be Qualifications. The class will have getters and setter methods for Captain, and the Captain class will push to persons class. This class will throw an error for an unqualified position.

The FirstOfficer class will take in one argument which will be Qualifications. The class will have getters and setter methods for FirstOfficer, and the FirstOfficer class will push to persons class. This class will throw an error for an unqualified position.

The Flight class will take in arguments and have attributes for taking in information and creating information (getters and setters) of flight number, aircraft type, captain, flight attendant, first officer, the origin airport, the destination airport, the scheduled takeoff time, the estimated takeoff time, and the actual touchdown time. It will push and pull to and from schedule manager. It will throw an error for trying to use an employee that is not located at the origin airport.

The FlightAttendant class will have getters and setter methods for flight attendants, and it will push to the persons class.

The FlightCancel will pass and take an argument from flight class (that it belongs to as a subclass) and schedule manager. It will have attributes for freeing up crew members to be used for other flights. It will pass back to the schedule manager class and flight class, which will pass to the persons class and crew positions.

The FlightChange class will pass and take an argument from flight class (that it belongs to as a subclass) and schedule manager. It will have action attributes for change of plane and pass back to schedule manager, plane, and airport. It will also pass information setters to Schedule manager for interface data.

FlightScheduled will pass and take an argument from flight class (that it belongs to as a subclass) and schedule manager. It will have getters and setters for the time and destination. It will show the scheduled flights information such as the estimated takeoff time and actual touchdown time. It will also pass information setters to Schedule manager for interface data.

The Person class will be dependent on and take in three arguments from the following: Captain, First Officer, and Flight Attendant. The class will assign getters and setters for each argument to set and pass along the information. It will have an attribute for change crew members.

Plane will have a type attribute and getters and setters of the two aircraft types: GBR-10 and NU-150. Plane will pass to airport and pull from schedule manager.

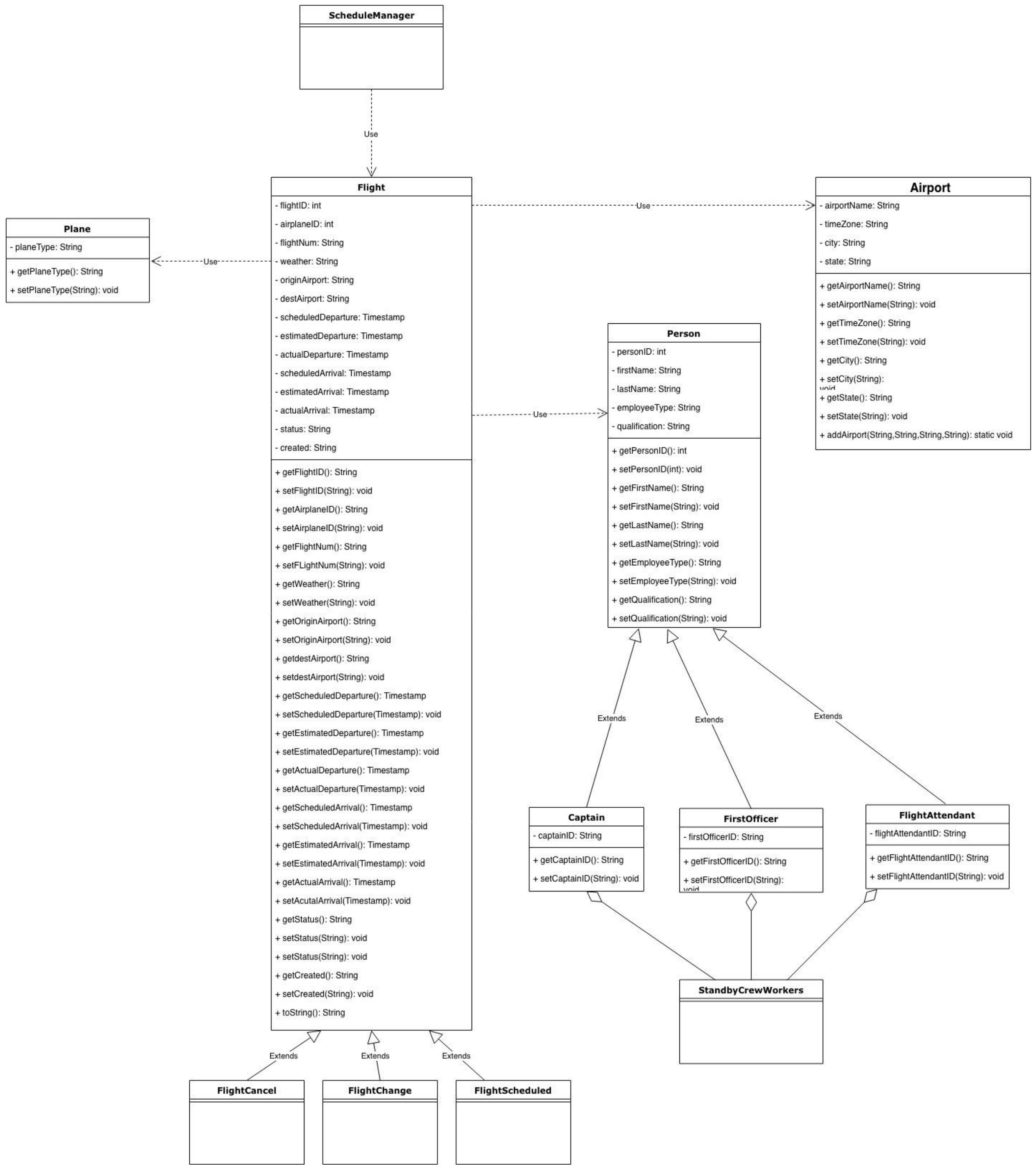
ScheduleManager takes in three arguments/dependencies: airport, plane, flight, and persons, and will push it to the database. Schedule Manager's attributes will require getters and setters via calls from the arguments in its class. Schedule Manager will have place attributes for crew members and aircraft. Schedule Manager will push additional data to user interface for flight records and keep an electronic log.

StandbyCrewWorkers will have getters and setters and will have separate attributes for each position: Captain, FirstOfficer and FlightAttendants, and for each aircraft type they're qualified for: GBR-10 and NU-150.

GLOSSARY

- CSE - Computer Science and Engineering (UNL department)
- SQL - Structured Query Language.

DETAILED DESIGN


ScheduleManager
Use
Flight
- flightID: int
- airplaneID: int
- flightNum: String
- weather: String
- originAirport: String
- destAirport: String
- scheduledDeparture: Timestamp
- estimatedDeparture: Timestamp
- actualDeparture: Timestamp
- scheduledArrival: Timestamp
- estimatedArrival: Timestamp
- actualArrival: Timestamp
- status: String
- created: String
+ getFlightID(): String
+ setFlightID(String): void
+ getAirplaneID(): String
+ setAirplaneID(String): void
+ getFlightNum(): String
+ setFLightNum(String): void
+ getWeather(): String
+ setWeather(String): void
+ getOriginAirport(): String
+ setOriginAirport(String): void
+ getdestAirport(): String
+ setdestAirport(String): void
+ getScheduledDeparture(): Timestamp
+ setScheduledDeparture(Timestamp): void
+ getEstimatedDeparture(): Timestamp
+ setEstimatedDeparture(Timestamp): void
+ getActualDeparture(): Timestamp
+ setActualDeparture(Timestamp): void
+ getScheduledArrival(): Timestamp
+ setScheduledArrival(Timestamp): void
+ getEstimatedArrival(): Timestamp
+ setEstimatedArrival(Timestamp): void
+ getActualArrival(): Timestamp
+ setAcutalArrival(Timestamp): void
+ getStatus(): String
+ setStatus(String): void
+ setStatus(String): void
+ getCreated(): String
+ setCreated(String): void
+ toString(): String
Plane
- planeType: String
+ getPlaneType(): String
+ setPlaneType(String): void
Use
Use
Airport
- airportName: String
- timeZone: String
- city: String
- state: String
+ getAirportName(): String
+ setAirportName(String): void
+ getTimeZone(): String
+ setTimeZone(String): void
+ getCity(): String
+ setCity(String): void
+ getState(): String
+ setState(String): void
+ addAirport(String,String,String,String): static void
Use
Person
- personID: int
- firstName: String
- lastName: String
- employeeType: String
- qualification: String
+ getPersonID(): int
+ setPersonID(int): void
+ getFirstName(): String
+ setFirstName(String): void
+ getLastName(): String
+ setLastName(String): void
+ getEmployeeType(): String
+ setEmployeeType(String): void
+ getQualification(): String
+ setQualification(String): void
Extends
Extends
Extends
Captain
- captainID: String
+ getCaptainID(): String
+ setCaptainID(String): void
FirstOfficer
- firstOfficerID: String
+ getFirstOfficerID(): String
+ setFirstOfficerID(String): void
FlightAttendant
- flightAttendantID: String
+ getFlightAttendantID(): String
+ setFlightAttendantID(String): void
StandbyCrewWorkers
Extends
Extends
Extends
FlightCancel
FlightChange
FlightScheduled