Georgia State University

Documentation of BAM Airlines Application

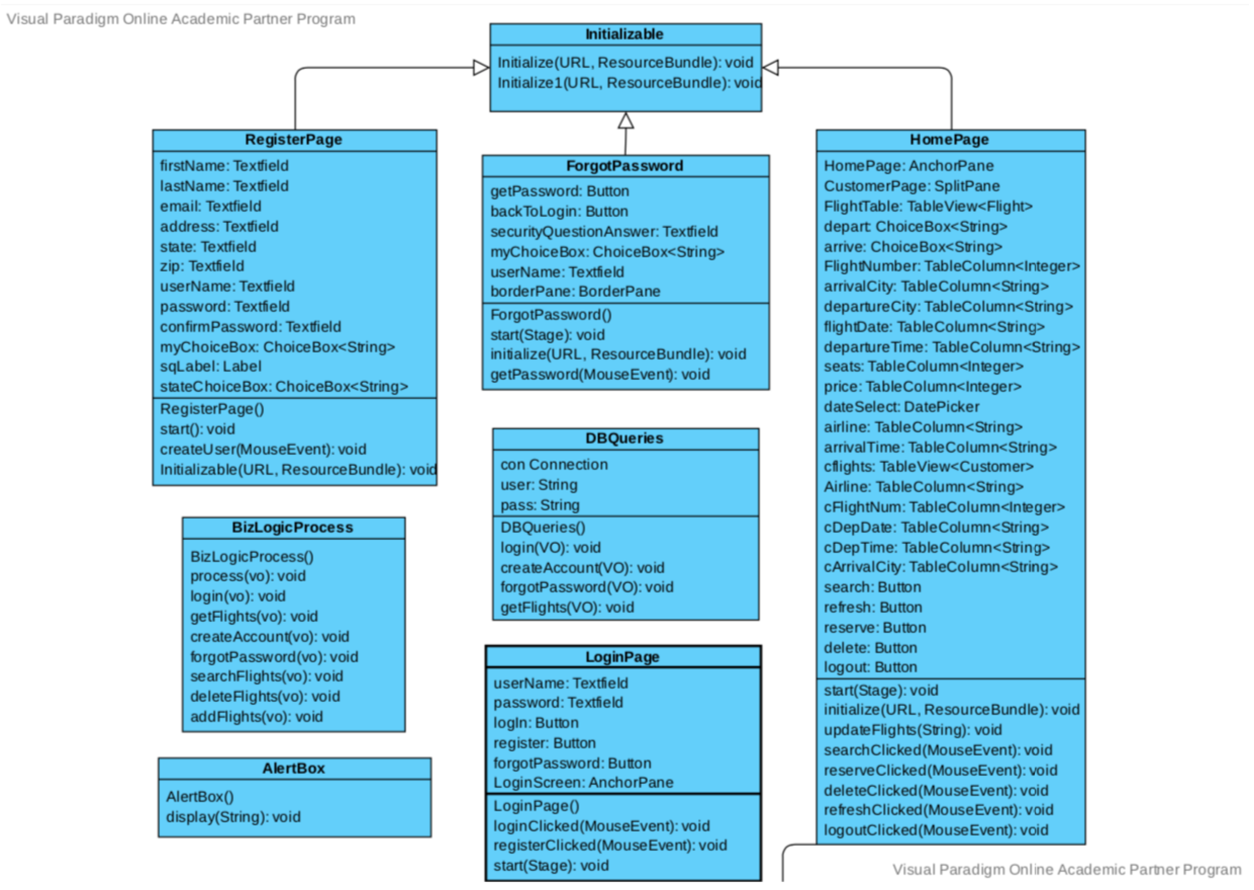
Andrew Barton
Biruk Larebo
Matthew Lee

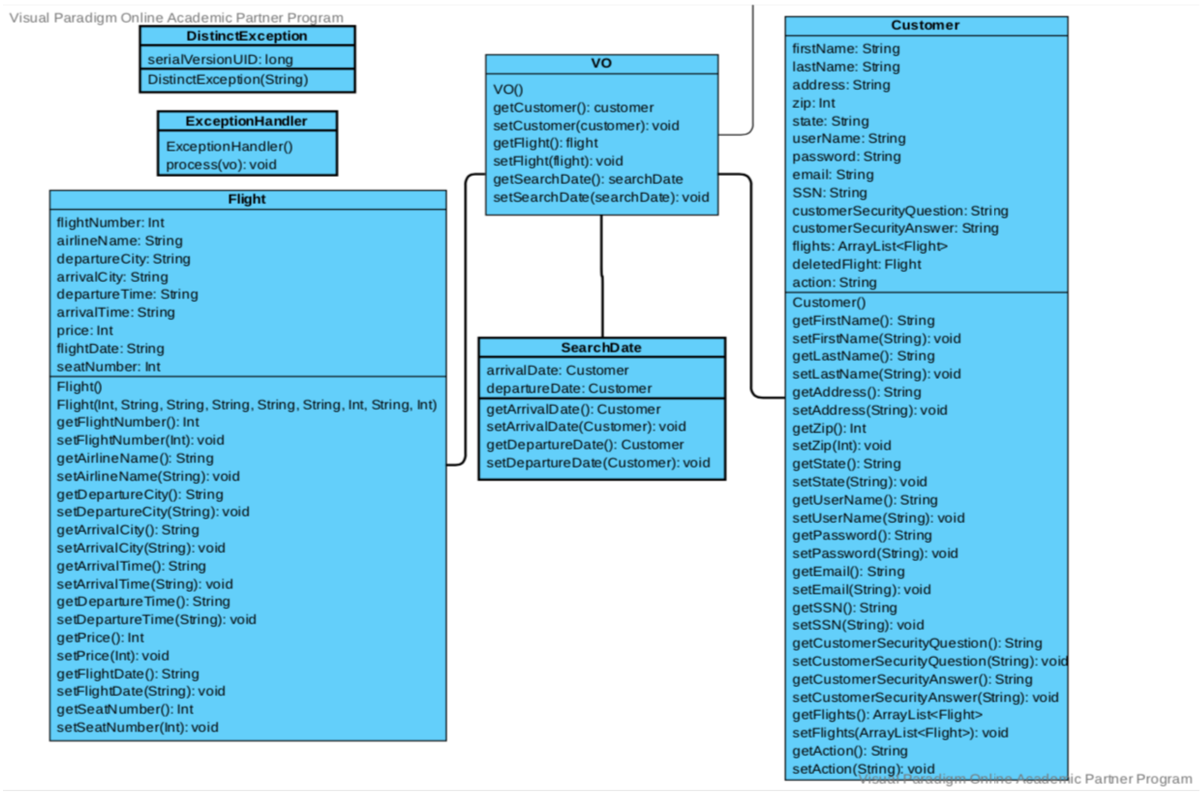
CIS 3270

User Requirements and Specifications

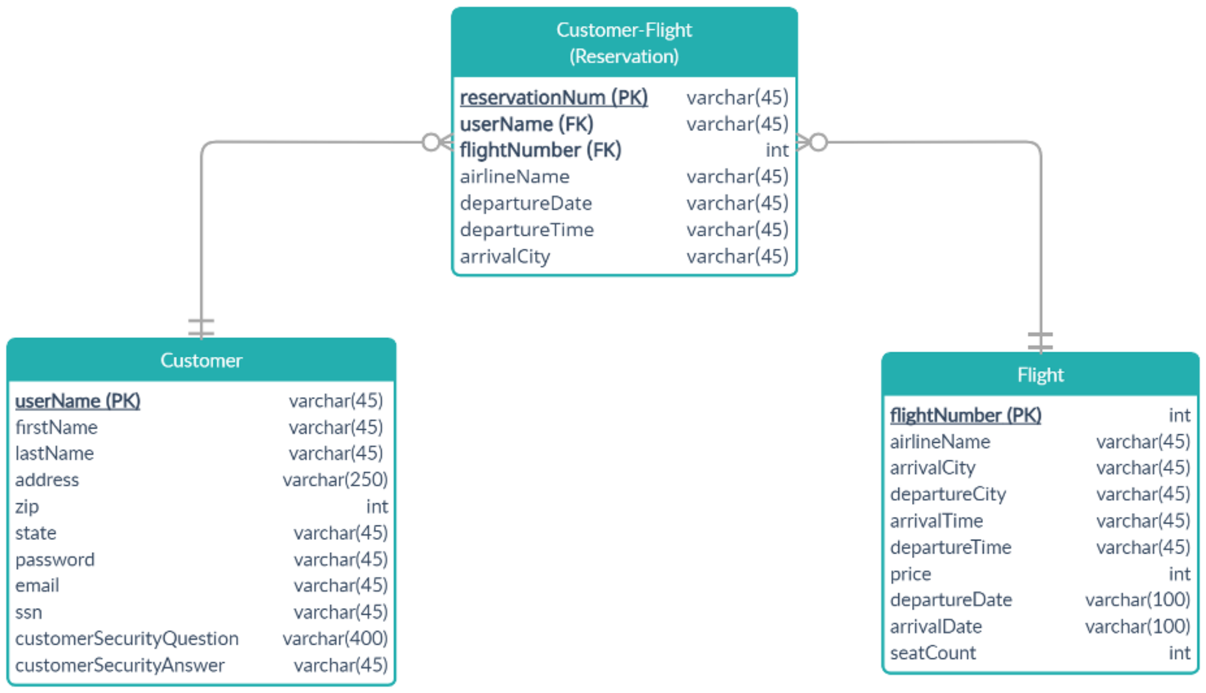
Our flight reservation application, “BAM Airlines,” is an application written in Java, meant to make it seamless for users to reserve flights. Upon the start of the application, the user is to see a login page. This page will give the user the option to either login, if they already have an account with BAM Airlines, or register with us if they don’t. Upon registration, the user must input information such as their first name, last name, address, zip code, state, username, password, email, social-security-number, and they must choose a password recovery question from the dropdown window and provide an answer with it. If the user has forgotten their password and therefore is unable to login, they are able to click the forgot password button and choose the security question that they answered when previously registering an account. If they provide the correct answer to the correlated security question, our program will provide them with the password that they previously registered with. Once they are in the home page of our application many options will become available to the user. First the user will set the parameters of the flight that they would like to reserve. They will select “Departing City” from a dropdown window, they will select a “Departing Date” and “Arrival Date” from two date pickers, and finally they will select a city that they would like to fly to by selecting from the “Arrival City” drop down arrow. After the parameters are selected, the user will click the “Search” button which will bring up any and all flights that match those parameters in our “Available Flights” table column. From there, the “Available Flights” table column will show the airlines, the flight number, the departure city, the arrival city, the departure time, the arrival time, the flight date, the number of available seats, and the price of the flight. The user is then to select a flight from the “Available Flights'' table column that meets their needs. Once a flight is selected the user is then to click the “Reserve” button to then add the chosen flight to the “Reserved Flights” table column at the top. The purpose of this table column is to store all of the flights that the user has reserved. In the “Reserved Flights” table column it will show the following attributes regarding the selected flight: airline, flight number, departure date, departure time, and arrival city. The user can then click the refresh page page button to see that their previously selected flight now has one less seat than it did before they reserved it. The user is now able to select any of their reserved flights and if they choose they can click the “Delete” button in order to remove that flight from their list of reserved flights. Finally, when the user is done, they are given the option to log out, by clicking the “LogOut” button in the top right of the UI. This button will take the user back to the login page.

Class Diagram


Visual Paradigm Online Academic Partner Program
Initializable
Initialize(URL, ResourceBundle): void
Initialize1(URL, ResourceBundle): void
RegisterPage
firstName: Textfield
lastName: Textfield
email: Textfield
address: Textfield
state: Textfield
zip: Textfield
userName: Textfield
password: Textfield
confirmPassword: Textfield
myChoiceBox: ChoiceBox<String>
sqLabel: Label
stateChoiceBox: ChoiceBox<String>
RegisterPage()
start(): void
createUser(MouseEvent): void
Initializable(URL, ResourceBundle): void
ForgotPassword
getPassword: Button
backToLogin: Button
securityQuestionAnswer: Textfield
myChoiceBox: ChoiceBox<String>
userName: Textfield
borderPane: BorderPane
ForgotPassword()
start(Stage): void
initialize(URL, ResourceBundle): void
getPassword(MouseEvent): void
HomePage
HomePage: AnchorPane
CustomerPage: SplitPane
FlightTable: TableView<Flight>
depart: ChoiceBox<String>
arrive: ChoiceBox<String>
FlightNumber: TableColumn<Integer>
arrivalCity: TableColumn<String>
departureCity: TableColumn<String>
flightDate: TableColumn<String>
departureTime: TableColumn<String>
seats: TableColumn<Integer>
price: TableColumn<Integer>
dateSelect: DatePicker
airline: TableColumn<String>
arrivalTime: TableColumn<String>
cflights: TableView<Customer>
Airline: TableColumn<String>
cFlightNum: TableColumn<Integer>
cDepDate: TableColumn<String>
cDepTime: TableColumn<String>
cArrivalCity: TableColumn<String>
search: Button
refresh: Button
reserve: Button
delete: Button
logout: Button
start(Stage): void
initialize(URL, ResourceBundle): void
updateFlights(String): void
searchClicked(MouseEvent): void
reserveClicked(MouseEvent): void
deleteClicked(MouseEvent): void
refreshClicked(MouseEvent): void
logoutClicked(MouseEvent): void
DBQueries
con Connection
user: String
pass: String
DBQueries()
login(VO): void
createAccount(VO): void
forgotPassword(VO): void
getFlights(VO): void
BizLogicProcess
BizLogicProcess()
process(vo): void
login(vo): void
getFlights(vo): void
createAccount(vo): void
forgotPassword(vo): void
searchFlights(vo): void
deleteFlights(vo): void
addFlights(vo): void
AlertBox
AlertBox()
display(String): void
LoginPage
userName: Textfield
password: Textfield
logIn: Button
register: Button
forgotPassword: Button
LoginScreen: AnchorPane
LoginPage()
loginClicked(MouseEvent): void
registerClicked(MouseEvent): void
start(Stage): void
Visual Paradigm Online Academic Partner Program


Visual Paradigm Online Academic Partner Program
DistinctException
serialVersionUID: long
DistinctException(String)
ExceptionHandler
ExceptionHandler()
process(vo): void
Flight
flightNumber: Int
airlineName: String
departureCity: String
arrivalCity: String
departureTime: String
arrivalTime: String
price: Int
flightDate: String
seatNumber: Int
Flight()
Flight(Int, String, String, String, String, String, Int, String, Int)
getFlightNumber(): Int
setFlightNumber(Int): void
getAirlineName(): String
setAirlineName(String): void
getDepartureCity(): String
setDepartureCity(String): void
getArrivalCity(): String
setArrivalCity(String): void
getArrivalTime(): String
setArrivalTime(String): void
getDepartureTime(): String
setDepartureTime(String): void
getPrice(): Int
setPrice(Int): void
getFlightDate(): String
setFlightDate(String): void
getSeatNumber(): Int
setSeatNumber(Int): void
VO
VO()
getCustomer(): customer
setCustomer(customer): void
getFlight(): flight
setFlight(flight): void
getSearchDate(): searchDate
setSearchDate(searchDate): void
SearchDate
arrivalDate: Customer
departureDate: Customer
getArrivalDate(): Customer
setArrivalDate(Customer): void
getDepartureDate(): Customer
setDepartureDate(Customer): void
Customer
firstName: String
lastName: String
address: String
zip: Int
state: String
userName: String
password: String
email: String
SSN: String
customerSecurityQuestion: String
customerSecurityAnswer: String
flights: ArrayList<Flight>
deletedFlight: Flight
action: String
Customer()
getFirstName(): String
setFirstName(String): void
getLastName(): String
setLastName(String): void
getAddress(): String
setAddress(String): void
getZip(): Int
setZip(Int): void
getState(): String
setState(String): void
getUserName(): String
setUserName(String): void
getPassword(): String
setPassword(String): void
getEmail(): String
setEmail(String): void
getSSN(): String
setSSN(String): void
getCustomerSecurityQuestion(): String
setCustomerSecurityQuestion(String): void
getCustomerSecurityAnswer(): String
setCustomerSecurityAnswer(String): void
getFlights(): ArrayList<Flight>
setFlights(ArrayList<Flight>): void
getAction(): String
setAction(String): void
Visual Paradigm Online Academic Partner Program

Data Model



Program Functionality and Flow

The process of running the application starts with our GUI. The very first thing the user will see is our login screen. From the login the user will input their username and password into the string text fields. In our program there will be a value object created with a customer inside that stores this input. We perform the setUsername and setPassword method to pass these values onto the Value Object. We will also pass a string message that will determine what process we are trying to do. In this instance the message that is passed is “Login”. The VO and String message will then get passed through our exception handler which will try the business logic process. The business logic process will then run a specific process depending on that message. In this case it is “login”so it will run the login process which then goes to our DBQueries, which will pass the VO. Once we go to the DBQueries, this is where in the specific login method, it will first run the getConnection() method from our ConnectDatabase class to connect to our database, then it will pass those values, extracting the customer from the VO (doing vo.getcustomer). Then it will run a query from our queries class which will get everything from our customer table and would get the username and password from the VO and check whether it exists in the database. If it does then it increases the variable count by 1 and returns true for the business logic process. This will then allow the user to go into the next page. If it is false and the count is still 0, the program will throw a distinct exception (our unique exception) which will then display on the application "Invalid Username or Password".

The rest of our program pretty much follows this pattern, where the first action performed is on the GUI, which will then move over to our exception handler which will try to pass that VO and String message through a process. From there the BizLogic will run the process associated with that String message that we passed earlier. From there it will go to the DBQueries which will connect to the database and run the relevant query and check regarding the object (customer or reservation) inside of the Value object and the information inside of the database. This will then execute the statement and if there is a match or the parameters/logic are correct, then it will perform the task such as logging in, searching flights, reserving flights,etc. Most of these tasks are all associated with the use of the username which is our unique identifier that allows us to save progress such as flights reserved. If the process is not successful it will throw an error, either using our distinct exception or an alert box that is triggered in the method. This is pretty much the flow of our program and the way that tasks are completed.