Technical Documentation

AUTOmatedPARK

Group 4

Joshua Beninson

Madhumitha Harishankar

Rashmi Loka

Shravanthi Muthuraman

Phu Phan

Daniel Selmon

AUTOmatedPARK

DEMO 2:

TECHNICAL DOCUMENTATION INDEX

Table of Contents

CODE INDEX

Data Access

DataAccess.cs: the file which links our classes to our database through SQL.

Entities

Customer.cs: class for a generic customer object. It defines the type of customer (be it a Walk-In, Reserved, or Contract), their billing information (if applicable), license plate# (if applicable), and assigned (if applicable). The applicability is determined by the customer type.

Floor.cs: class to define a floor in the garage in terms of the numbers of spots, available spots, whether or not the floor is fully occupied, and the floor number.

Garage.cs: class to define the number of floors (instantiated by Floor.cs) that should be associated with a Garage object in terms of its private element called floorArray.

RegisteredCustomer.cs: derived class from Customer which encompasses additional elements for a registered customer (userName, password, and a linked list of reservations called reservationList).

Reservation.cs: class that defines reservations in terms of start and end times, as well as duration. They are stored in linked lists that are associated with the RegisteredCustomer class.

Spot.cs: class that defines Spots that are associated with a Floor which is in turn associated with a Garage. It keeps track of vacancy, as well as the Customer that might occupy it.

Database Stored Procedures:

getSpots: This procedure is used to get the nunmber of spots in each floor of the garage. It takes no input, and looks into the garage table and reads the singular value from the column called "spots" and returns this. This procedure modifies the tables member, member_address, and member_login. Variables were selected carefully to be self-explanatory.

getFloor: /*This procedure is used to get the number of floors in the garage. It takes no input, and looks into the garage table and reads the singular value from the column called "floors" and returns this. This procedure modifies the tables member, member_address, and member_login. Variables were selected carefully to be self-explanatory. */

setGarage: /*This procedure is used to set the number of floors and spots in the garage when the admin first logs into the system. The garage is configured once this procedure is called once. It takes as input the number of floors and spots in each floor of the garage. It modifies the table called Garage. Variables were selected carefully to be self-explanatory. */

getOccupancy: /*This procedure is used to return the status of occupancy of a spot from the reserve_parking table. It takes as input a spot ID, and returns an integer. A value of 1 indicates that the spot is occupied. A value of 2 indicates that it is overstaying, and a value of 3 indicates that the spot is a noshow. It procedure does not modify any table. Variables were selected carefully to be self-explanatory.*/

setOccupancy: /*This procedure is used to set the status of occupancy of a spot. It takes as input a spot ID, and an integer that indicates the occupancy status. A value of 1 indicates that the spot is occupied. A value of 2 indicates that it is overstaying, and a value of 3 indicates that the spot is a noshow. It procedure does not modify any table. Variables were selected carefully to be self-explanatory.*/

setLogHistory: /*This procedure is used to add a log of reservation to the logHistory. It takes no input and neither returns anything. This procedure, when called, looks into the reserve_parking_online table, and adds expired reservations to logHistory table, and then delets these expired reservations from the running table reserve_parking_online. This procedure modifies reserve_parking_online and logHistory tables. Variable names were chosen to be self-explanatory.*/

getLogHistory: /*This procedure is used to return the past garage usage of a member. It takes as input the memberid and returns the past garage usage history in SqlDataReader which is then parsed into an array. This procedure modifies no table. Variables were selected carefully to be self-explanatory.*/

addNewCustomer: This procedure is used to create a new member during registration. A member id is assigned to each user. The values provided by the user while registering is fed as input to this procedure. The table fields are set to the provided input values. This procedure modifies the table's member, member_address, and member_login. Variables were selected carefully to be self-explanatory.

addReservationToDatabase: This procedure is used to create a reservation for a registered customer (has a member id). This assigns a reservation id. One of the inputs—member id—should be extracted from the database. The other inputs such as reservation start and end date time should be the values provided by the user. This procedure modifies the reserve_parking_online table. Variables were selected carefully to be self-explanatory.

create_member: This procedure is used to add a new member to the database. It takes in as input member registration details and modifies member, member_login and member_address tables to put in the data. The tables have foreign keys and constraints set so that each memberid from member is associated with the ID in member_login and member_address. More than one member with the same details is not allowed, and upon trying to input more than 2 same members, the procedure throws an error. This procedure modifies the table member, member_login and member_address. Variable names were carefully selected to be self-explanatory.

deleteReservationFromDatabase: This procedure is used to cancel a reservation made by a registered reserved user. One of the inputs, member id should be extracted from the database. The other inputs such as the start and end date time of the reservation that needs to be cancelled should be the values provided by the user. This procedure modifies the reserve_parking_online table. Variables were selected carefully to be self-explanatory.

getAmountDue: This procedure is used to retrieve the amount owed by a registered customer at any given time. This takes input of member id. This procedure looks up the Sales_Account table to retrieve the required information. Variables were selected carefully to be self-explanatory.

getMemberDetails: This procedure is used to retrieve details of a member given the license plate number associated with him. This procedure takes as input a license plate number. It then does a lookup inside table reserve_parking_online and searches it for the row whose carPlateNo matches the input given; this row is returned back. This procedure does not modify any table. Variables were selected carefully to be self-explanatory.

getMemberDetailswithUsername: This procedure is used to retrieve details of a member given a string associated with the member's username.It first searches for a row in member_login table where the userName = input string.Once this row it found, it looks up the membered of the member returned, and uses this ID to do a lookup in reserve_parking_online table . It returns the row containing this memberid, and hence the user details are retrieved. Note: We call a procedure verifyIsUser that verifies if a username and password match is found for an input. Only after this check passes is this procedure called. That is why the password string is not used in this procedure. This procedure does not modify any table. Variables were selected carefully to be self-explanatory

checkIfUser: This procedure is used to validate a username/password pair. The username and password are given as input and it scans the member_login table to see if this pair is present there. If yes, it returns a count of it. Hence, if the count is 1, then the user exists, else this is an invalid username/password combination. This procedure does not modify any table. Variables were selected carefully to be self-explanatory.

Website:

Login.aspx: Default webpage that gets display, asking for user login for entry into the system.

Home.aspx : Home page that greets a user after successful login.

MakeReservation.aspx: Allows a user to make reservations, and displays options for both monthly contracts and one-time reservations.

ManageReservation.aspx: Allows a user to edit or cancel a reservation previously made.

MyAccount.aspx: Allows a user to view all his past account details, including, money owed, billing details, license plate numbers, past garage usage history.

ContactUs.aspx: Displays the address and contact details of AutoMatedPark.

AboutUs.aspx: Gives a brief overview of who we are and what we do.

Policies.aspx: Details the policies that a user must read and agree to comply with before making reservations.

Website add-ons: Great use of AJAX toolkit and jQUERY to aid in enhancing UI to provide a great user experience.