Automated Parking Garage

Team Name: Team 2

Team Members:
- Johnnie Mares
- Valerie Ruvalcaba
- Derek Zhang
- Dat Pham
- David DeGirolamo
- Sean Sidwell

Preface:

Version 3

The contents of this report cover the details of the automated parking garage project that has been in development for four months now. The effort made by the production team was met with difficulties and obstacles involving exactly how to implement the parking garage. After careful planning and testing the team is fully confident in the product's implementation. This report aims to cover the architectural design changes made to the system since the last report, as well as detailed descriptions of said changes as to why they were made. Clients as well as others who wish to implement this product may use this documentation as a reference to features present in the current/live instance of the automated parking garage. Requirement changes will also be discussed and the rationale behind these decisions.

Why this report?

Engineers who read this report will be able to understand in more detail the inner workings of the system, including functions calls, interface interactions, and system storage. This report will also include uses of outside libraries and their involvement in the parking garage system.

Who is this report for?

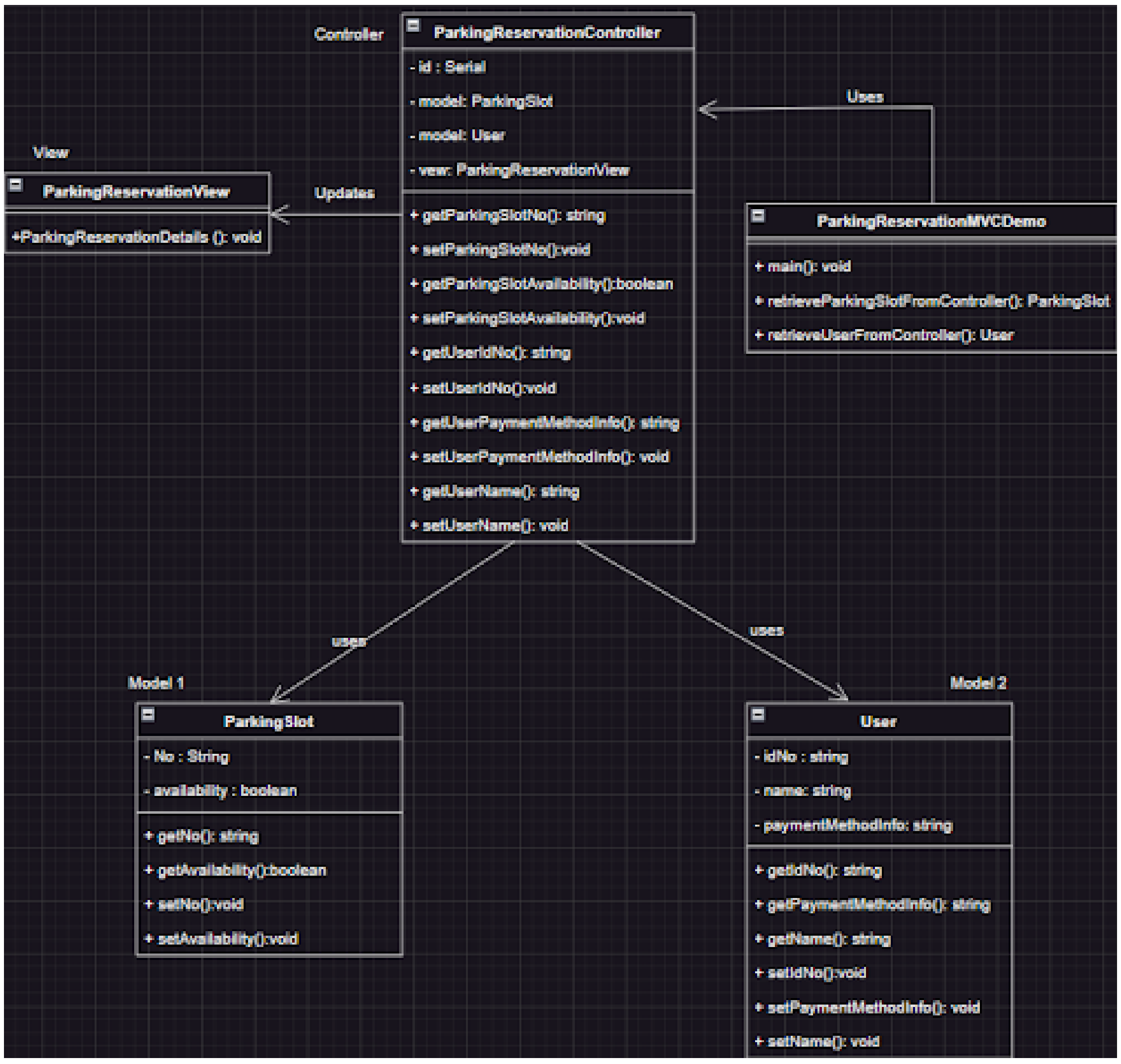
This report is for companies / engineers who wish to implement an automated parking structure system for parking lot structures. This report offers a detailed understanding of the development cycle of an automated parking structure. Readers who are interested in learning about the logic behind the garages implementation will find this report most insightful.

Introduction:

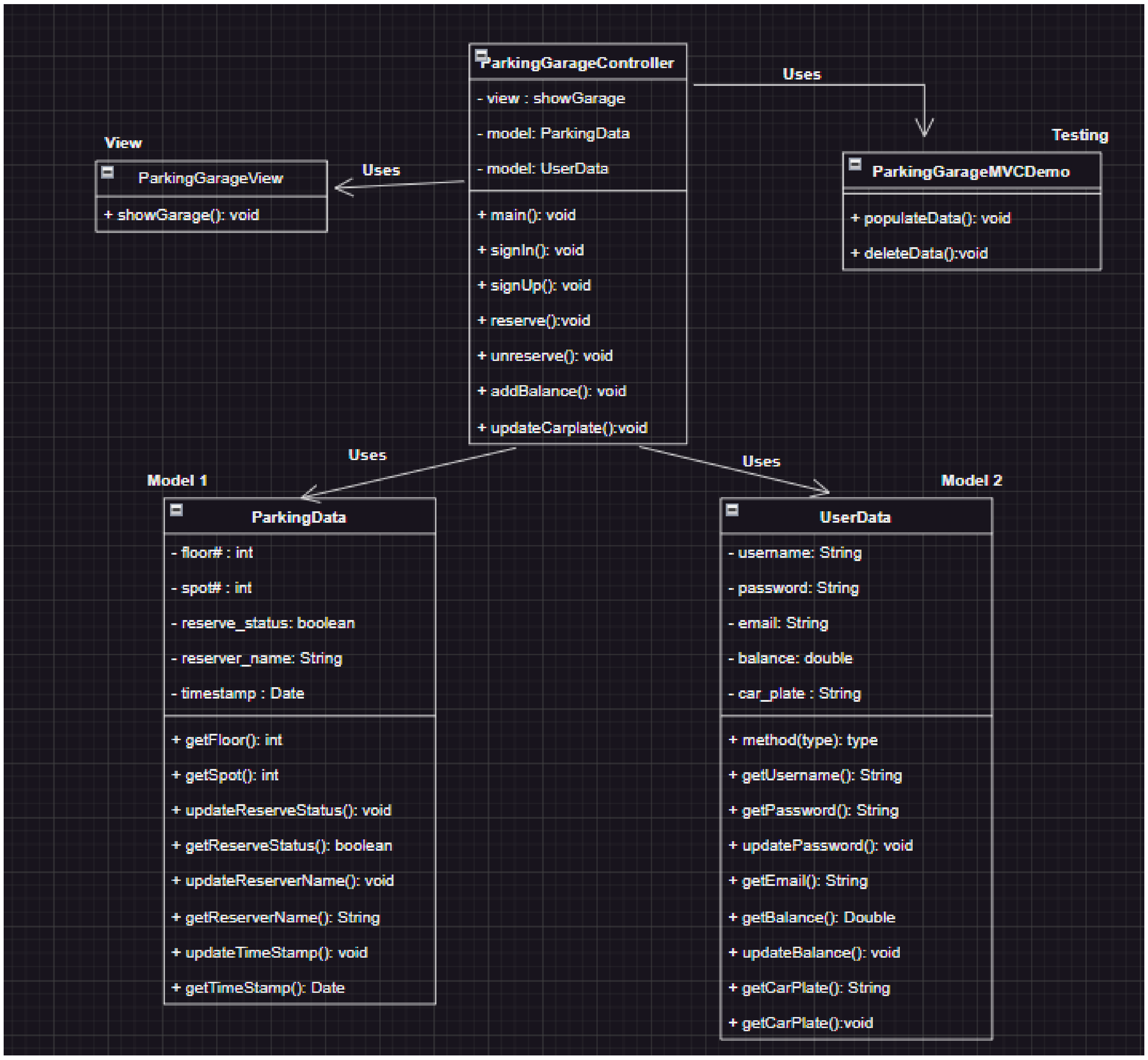
Project Need: the project arises from the need to handle/manage a parking structure automatically without the need for actual on site management. The Automated parking garage system serves a liaison between the drivers (users) and the clients business parking structures. The client business who commissions this product will be able to have a system that manages traffic going in and out of their buildings while producing a steady flow of income. The parking garage will distribute permits to users who wish to use the lot, the system will keep track of what spots are open and which spots are full / reserved. The system will also allow users to create an account and sign up in order to resolve a spot within the allotted time. The parking garage will not permit entry if there is no vacant lot, letting the client company no need to worry about the lot overflowing.

Architectural Changes:

Previous Architect


Controller
ParkingReservationController
- id : Serial
- model: ParkingSlot
- model: User
- vew: ParkingReservationView
+ getParkingSlotNo(): string
+ setParkingSlotNo():void
+ getParkingSlotAvailability():boolean
+ setParkingSlotAvailability():void
+ getUserIdNo(): string
+ setUserIdNo():void
+ getUserPaymentMethodInfo(): string
+ setUserPaymentMethodInfo(): void
+ getUserName(): string
+ setUserName(): void
Uses
View
ParkingReservationView
+ParkingReservationDetails (): void
Updates
ParkingReservationMVCDemo
+ main(): void
+ retrieveParkingSlotFromController(): ParkingSlot
+ retrieveUserFromController(): User
uses
uses
Model 1
ParkingSlot
- No : String
- availability : boolean
+ getNo(): string
+ getAvailability():boolean
+ setNo():void
+ setAvailability():void
Model 2
User
- idNo : string
- name: string
- paymentMethodInfo: string
+ getIdNo(): string
+ getPaymentMethodInfo(): string
+ getName(): string
+ setIdNo():void
+ setPaymentMethodInfo(): void
+ setName(): void

Updated Architect



1. In our updated architectural version. We still mostly keep the majority of the attribute and the function but we changed majority of the names so it will be easier to understand (ex: availability in ParkingSlot to reserve_status in ParkingData, No in ParkingSlot to floor# and spot# in ParkingData, etc)
2. We added more attributes such as *timestamp* in ParkingData that is used to calculate the Parking Spot charging fee when users decide to unreserve, or *car_plate* (or license plate) in UserData model that is used to keep track of which user is currently own which vehicle and to keep track of which reserved spot belong to which car, and *balance* in UserData model is used pay for the reservation when they decide leaving the ParkingGarage (unreserve), etc

3. In our previous architecture, we planned the ParkingReservationMVCDemo table is technically our main file using the main function and the other functions (retrieveParkingSlotFromController and retrieveUserFromController) is used for testing to see we are able to get data from ParkingSlot and User model. However, we saw these functions do not serve its purpose much. So we updated where in updated Architect, ParkingGarageController is now our main file that inherits all functionalities of other models (ParkingData, UserData, ParkingGarageView, and ParkingGarageMVCDemo) to create a UI and functions (signIn, signUp, reserve, etc) to meet the requirements. Where in ParkingGarageMVCDemo, the populate function is used to populate some test data to the parkingData and userData model in the database that can be used for testing. And the deleleteData function is used to delete all currently existing data in ParkingData and UserData models in the database for resetting the testing data.

Detailed Design Change:

We made many detailed changes over the course of our project.

- Added Lot name variable when creating a new Lot
  - Creates more organization and convenience with keeping track of lot or multiple lots
- Added function to unreserve a reservation
  - In case of a change in plans for user or reservation is no longer needed
- Instead of having a class (in ParkingGarage(ver2).py) with set and get methods we can create a schema that takes username, password, email, license plate #, and balance as input
  - Using a schema create better organization in keeping track of reservations
- Added function to log out
  - So that user doesn't stay signed in when not using system
- Added function to delete account
  - So that if user no longer needs account they can delete it
- Added ShowGarage(), which displays spaces that are occupied/unoccupied
  - Makes it easier to see which spaces are available/unavailable
- Added color variable to label the spaces in the lot even further. If the space is red it is occupied and if blue unoccupied
  - Creates further detailed labeling for each spot to show if available/unavailable
- Added LeavingLot(), where user input floor number and space number to let system know that space is no longer needed and can become available, changing the color to blue
  - Helps update system when occupied space becomes available
- When adding parking data to schema, reserver name variable added

    - For further clarification of reservation details
- Added function to validate reserver’s name and when validated displays current reserved spaces in lot
    - Keeps user updated of current reservation(s)
- Added electric parking with green labeled spaces when unoccupied and red when occupied
    - Now have space available for electric cars that need to charge their battery during reservation
- Added parking is full function along with a available spot counter variable and a max spaces variable to compare it to
    - Helps system know when a lot is full/not full
- Added function where if lot is fully reserved, user is not able to reserve a space in that lot
    - A user should not be able to make a reservation at a lot if it is fully reserved
- Added timestamp and rate for reservation
    - Keeps track of how long reservation is made for and how much it will cost user
- Added restriction to one parking spot per user.
    - Because each user can only have one license plate register to their account. So having the user able to reserve more than one parking spot is making no sense.
- Added Web Application to keep track of users, reservations in the lot, and remotely register users.
    - This is to facilitate better management of users, and provide a base template on which further online integration with the terminal application would take place.

Requirements Change:

Feature Removed	Reason	Note
Ability for managers to check busiest times	Manager accounts with special permissions would need to have been implemented, causing complexities with authorization and authentication beyond the scope of the project.	This was changed to managers being able to access user info and parking lot data remotely through a separate online service
Placing temporary reservation on parking spots when added to cart	The implementation of a cart feature was removed due complexities dealing with transactions, as well as restrictions placed on the number of spaces a user is allowed to hold. As a result, all features associated with the cart have been removed.	
Locking accounts after incorrect password input	Locking accounts at the terminal program level of implementation would be counter-productive toward the client’s intended purposes, as it would bar users	

	from accessing and enacting transactions.	
Transaction system	Removed implementation due to concerns with complexity and security. Standard procedure would be to outsource this to a third party application (such as TransAct with CSULB's payment system). Beyond the scope of our project. However, if this program is actually being used in real life in the future. This feature will definitely need to be implemented.	

The overall architecture of our project did not change too much, but certain requirements (as listed above) were changed or removed in order to set implementation standards under the scope of this assignment. The overall architecture stayed the same due to the majority of requirement changes being functionalities in existing systems being altered or removed without the removal of the system itself.