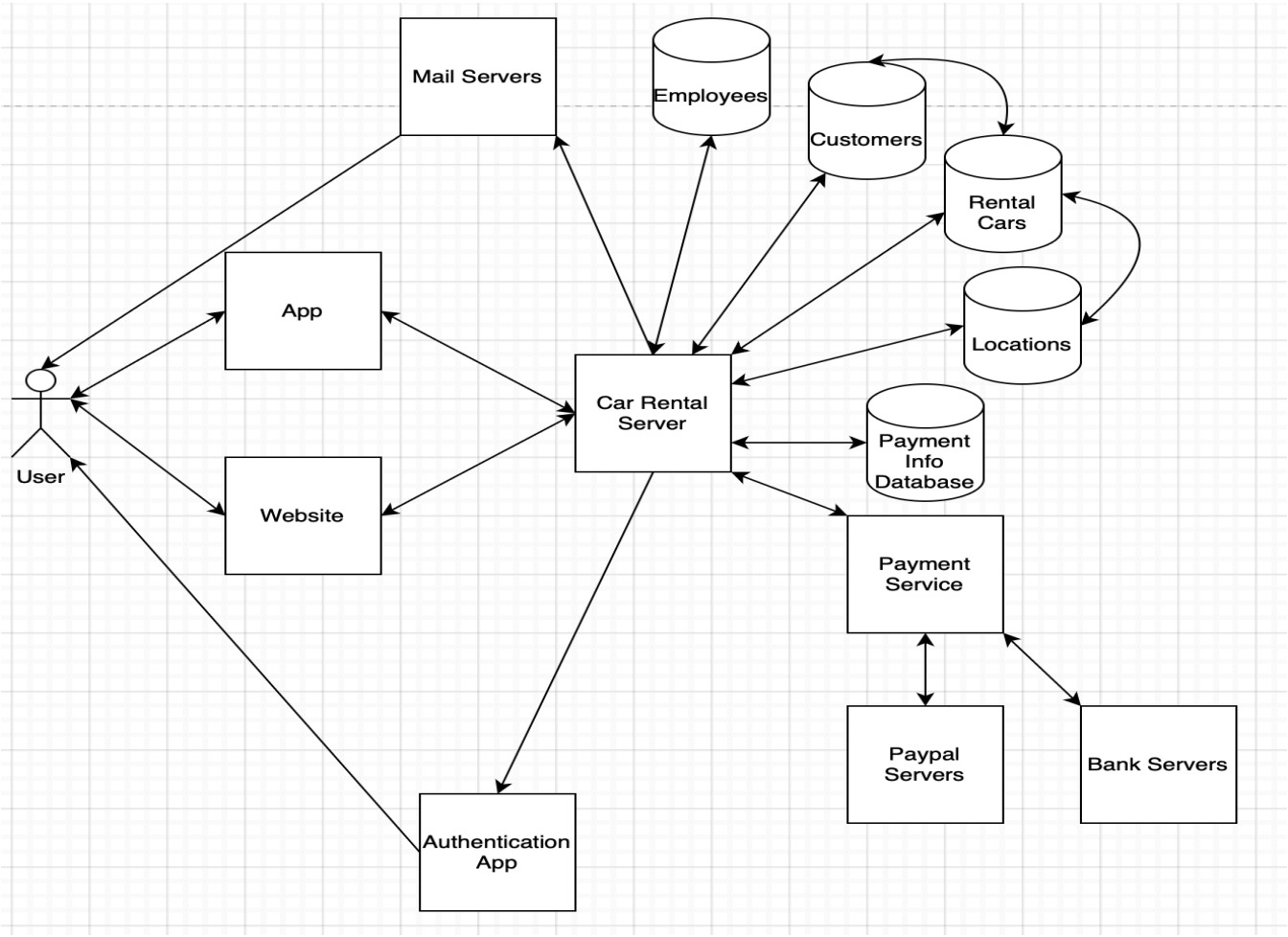
Software: Car Rental System
Team Members: Anthony Do & Ryan Brodey

System Description: The objective behind the development of BeAvis's new car rental software system is to create a streamlined and user-friendly alternative to the traditional pen-and-paper rental process. This initiative aims to modernize BeAvis's rental operations, enhancing efficiency and elevating the overall customer experience. The system will encompass essential functionalities required for effective management of a car rental company, including account management, rental agreement handling, and providing efficient tools for employee tasks. Both customers and employees stand to benefit from this system, with customers enjoying a simplified interface, and employees being able to seamlessly update and manage the system as needed. This document serves as a comprehensive guide for the implementation of the car rental software system, covering user requirements, system specifications, and other pertinent details crucial for its successful deployment.

Software Architecture Overview:

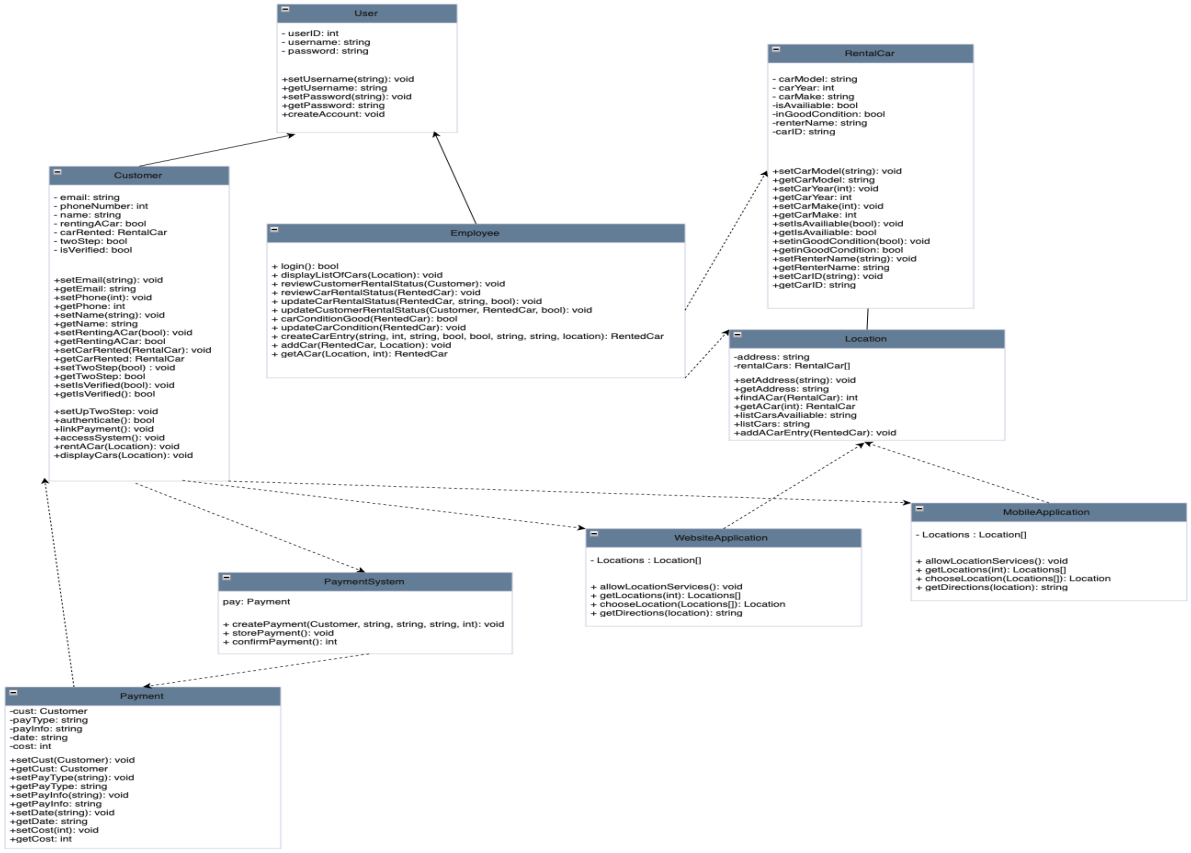
Architectural Diagram:



Description of Architectural Diagram:

In this diagram, the user interacts with the car rental server either through the mobile app, or the website. The Car Rental Server also interacts with mail servers, the employee database, the customer database, the rental car database, the locations database, the payment info database, the payment service, and the authentication app. Both the mail servers and authentication app interact with the user by helping to verify the status of their account and confirming if that is their account. The employee database holds the accounts that have employee access. The customer database holds the accounts with customer level access. The rental car database holds the data for the various cars up for rent. The locations database holds the data for the various branches of the rental company. The payment info database will hold only the data on the payment info of each transaction made to rent a car. For the payment service, it will interact with paypal servers and bank servers in order to insure that payment is made to the car rental company.

UML Diagram:



Description of Classes and Attributes:

User:

- Description: The User class is a class that will be the general structure that the Employee and Customer class will inherit from and branch from.
- Attributes:
  - userID: int - A unique identifier assigned to each user.
  - username: string - the user's account's name.
  - password: string - The password used for accessing the user's account.
- Operations:
  - setUsername(string): void - sets the username attribute to a provided string
  - getUsername: string - returns the value of the username attribute
  - setPassword(string): void - sets the password attribute to a provided string
  - getPassword: string - returns the value of the password attribute
  - createAccount(): void - creates/save/stores a user object into the database

Employee:

- Description: This class represents employees of the car rental company. In addition, this class inherits the attributes and the operations of the user class.
- Operations:
  - login(): bool - This method verifies the employee's login credentials.
  - displayListOfCars(Location): void - This operation takes a provided location data input and displays a list of the cars at that location.
  - reviewCustomerRentalStatus(Customer): void - This function retrieves and displays the rental status for a specific customer.
  - reviewCarRentalStatus(RentedCar): void - This function retrieves and displays the rental status for a specific car.
  - updateCarRentalStatus(RentedCar, string, bool): void - This method updates the rental status of a car within the system.
  - updateCustomerRentalStatus(Customer, RentedCar, bool): void - This method updates the rental status of a customer within the system.
  - carConditionGood(RentedCar): bool - returns whether a provided RentedCar data type is in good condition. True equal good. False equals bad.
  - updateCarCondition(RentedCar): void - Updates the condition of the provided RentedCar data type.
  - createCarEntry(string, int, string, bool, bool, string, string, location): RentedCar - This method creates a new RentedCar object.
  - addCar(RentedCar, Location): void - This method adds/stores a provided RentedCar data type to the array of RentedCars in the provided Location data type.
  - getACar(Location): RentedCar - This operation returns a RentedCar data type with the specified carID in the array of RentedCars in the provided Location data type.

Customer:

- Description: This class represents the customers that come to the car rental company. In addition, this class inherits the attributes and the operations of the user class.
- Attributes:
    - email: string - The email address associated with the user's account.
    - phoneNumber: int - The customer's phone number without hyphens
    - name: string - The full legal name of the user.
    - rentingACar: bool - Boolean of whether the customer has rented a car. True means yes, they are renting a car. False means no, they are NOT renting a car.
    - carRented: RentalCar - The car the customer is currently renting. If the customer is NOT renting a car, then the value is NULL.
    - twoStep: bool - Illustrates if the customer has two-step authentication. True equals yes, and False equals no.
    - isVerified: bool - A boolean flag indicating whether the user's account has been verified.
- Operations:
    - setEmail(string): void - Sets the email address of the user to the specified string.
    - getEmail: string - Retrieves user's email address.
    - setPhone(int): void - Sets the phone number of user to a provided integer
    - getPhone: int - Retrieves user's phone number
    - setName(string): void - Sets name of the user to specified string
    - getName: string - Retrieves the user's name
    - setRentingACar(bool): void - Sets status that indicates whether the user is currently renting a car
    - getRentingACar: bool - Retrieves status showing whether a user is currently renting a car.
    - setCarRented(RentalCar): void - Sets the car rented by user to provided RentalCar object.
    - getCarRented: RentalCar - Retrieves RentalCar object that represents the car rented by the user.
    - setTwoStep(bool) : void - Sets the status indicating if two-step authentication is enabled for the user's account.
    - getTwoStep: bool - Retrieves the status that indicates if two-step authentication is enabled for the user account.
    - setIsVerified(bool): void - Sets the status that indicates whether the user has a verified account.
    - getIsVerified(): bool - Retrieves the status that indicates whether the user has a verified account.
    -
    - setUpTwoStep: void - Initiates the process to set up two step authentication with the user
    - authenticate(): bool - Pings and links to customer's authentication app so that the user can authenticate themselves with that app.

- linkPayment(): void - Initiates the payment process for renting a car
- accessSystem(): void - Connects the customer to the system interface either the website or the mobile app.
- rentACar(Location): void - Initiates the process of renting a car at the provided location of where the customer is renting from
- displayCars(Location): void - Displays a list of the available cars at the provided location.

MobileApplication:

- Description: This class represents the mobile application interface utilized for accessing the car rental system.
- Attributes:
  - Locations: Location[] - list of all the locations of the car rental company
- Operations:
  - allowLocationServices(): void - This function activates location services on the mobile device.
  - getLocations(int): Locations[] - Retrieves a list of locations based on whatever criteria the user specifies.
  - chooseLocation(Locations[]): Location - Allows the user to choose a location from a list of provided locations.
  - getDirections(location): string - Retrieves directions from user's current location to the destination that is specified.

WebsiteApplication:

- Description: This class embodies the website interface utilized for accessing the car rental system.
- Attributes:
  - Locations: Location[] - list of all the locations of the car rental company
- Operations:
  - allowLocationServices(): void - This function activates location services on the mobile device.
  - getLocations(int): Locations[] - Retrieves a list of locations based on whatever criteria the user specifies.
  - chooseLocation(Locations[]): Location - Allows the user to choose a location from a list of provided locations.
  - getDirections(location): string - Retrieves directions from user's current location to the destination that is specified.

RentalCar:

- Description: This class encapsulates the core functionalities of the rental system.
- Attributes:
  - carModel: string - Represents the model of the car.
  - carYear: int - Year of the rental car.

        - carMake: string - Make or manufacturer of the rental car.
        - isAvailiable: bool - Indicates if the rental car is available.
        - inGoodCondition: bool - Indicates if the rental car is in good condition to be rented.
        - renterName: string - Name of the renter who has rented the car.
        - carID: string - Identifier assigned to the rental car.
    - Operations:
        - setCarModel(string): void - Sets the rental car model to string.
        - getCarModel: string - Gets rental car's model.
        - setCarYear(int): void - Sets year of manufacture of rental car to integer.
        - getCarYear: int - Gets the year of manufacture of the rental car.
        - setCarMake(int): void - Sets the make or manufacturer of the rental car.
        - getCarMake: int - Gets make or manufacturer of the rental car.
        - setIsAvailiable(bool): void - Sets availability status of the rental car.
        - getIsAvailiable: bool - Gets availability status of the car.
        - setinGoodCondition(bool): void - Sets condition of the rental car.
        - getinGoodCondition: bool - Sets the condition status of the rental car.
        - setRenterName(string): void - Sets the name of the renter of the car.
        - getRenterName: string - Gets name of the renter of the car.
        - setCarID(string): void - Sets the identifier of the rental car.
        - getCarID: string - Gets identifier of the rental car.

Location:

- Description: This class represents a location of the car rental company.
- Attributes:
    - address: string - the address of where the location is
    - rentalCars: RentalCar[] - an array of the RentalCar at this location
- Operations:
    - setAddress(string): void - set the value of address to the provided string
    - getAddress: string - returns the the value of address
    - findACar(RentalCar): int - returns an integer, which is the index of the RentalCar data type in the RentalCar array, rentalCars
    - getACar(int): RentalCar - returns a RentalCar data type in the rentalCars array given an integer value which is the index
    - listCarsAvailiable: string - displays a list of the cars available to be rented
    - listCars: string - displays a list of all the cars including those unavailable to be rented
    - addACarEntry(RentedCar): void - adds a provided RentalCar data type to the RentalCar array

Payment:

- Description: This class represents the transaction to rent a car.
- Attributes:
    - cust: Customer - the customer wanting to rent the car

        - payType: string - the type of payment
        - payInfo: string - info relating to the payment type
        - date: string - the date of the transaction
        - cost: int - the cost of the transaction
    - Operations:
        - setCust(Customer): void - sets the value of cust with a provided Customer data type
        - getCust: Customer - returns the value of cust
        - setPayType(string): void - sets the value of payType with the provided string
        - getPayType: string - returns the value of payType
        - setPayInfo(string): void - sets the value of payInfo with the provided string
        - getPayInfo: string - returns the value of payInfo
        - setDate(string): void - sets the value of date with the provided string
        - getDate: string - returns the value of date
        - setCost(int): void - sets the value of cost to the provided int
        - getCost: int - returns the value of cost

PaymentSystem:

- Description: This class represents the system responsible for managing payment-related information.
- Attributes:
    - pay: Payment - contains all the information relating to the transaction of renting a car
- Operations:
    - createPayment(Customer, string, string, string, int): void - Starts the process of creating a transaction that is for a customer's rental.
    - storePayment(): void - Stores the user's payment information in the system after a transaction.
    - confirmPayment(): int - Confirms the transaction and returns the cost of renting the car.

Development Plan & Timeline:

Partitioning of Tasks:

RentalCar

- Completed as soon as possible
- Ryan Brodey

Location

- Complete 2 weeks after RentalCar class

- Ryan Brodey

WebsiteApplication
- Start front-end of website as soon as possible
    - 2 months at most
- Within 3 months finish back-end
- Anthony Do & Ryan Brodey

MobileApplication
- Start front-end of app after website is completed
- Complete within 5 months after starting
    - Follow as Website Application
        - 2 months front-end
        - 3 months back-end
- Anthony Do & Ryan Brodey

User
- Complete as soon as possible
- Anthony Do

Customer
- Complete 2 weeks after user class is completed
- Anthony Do

Employee
- Also 2 weeks to complete after customer class is completed
- Anthony Do

Payment
- Can only be started after all other classes except WebsiteApplication and MobileApplication are completed
- 1 week is given to complete this class.
- Ryan Brodey

PaymentSystem
- Only after payment class is completed, begin implementation
- 3 weeks given to completed this class
- Anthony Do & Ryan Brodey

Team Member Responsibilities:
- Ryan Brodey - will be in charge of implementing the RentalCar, Location, and Payment classes.
- Anthony Do - will be responsible for completing the User, Customer, and Employee classes.
- Together - both will need to work together to finish the WebsiteApplication, MobileApplication, and the PaymentSystem classes

Final Notes:

- Overall time given to complete this system should be more or less than 6 or 7 months.

Verification Test Plan:

Unit Tests:

Unit 1: getUsername() : string

Case 1:

User a;
a.userID = 1
a.username = “abc”
a.password = “2”
If a.getUsername() == “abc”
    Return PASS
Else
    Return FAIL

Case 2:

User a;
a.userID = 1
a.username = “123”
a.password = “2”
If a.getUsername() == “123”
    Return PASS
Else
    Return FAIL

Case 3:

User a;
a.userID = 1
a.username = “#$@”
a.password = “2”
If a.getUsername() == “#$@”
    Return PASS
Else
    Return FAIL

Unit 2: getPassword() : string

Case 1:

User a;
a.userID = 1
a.username = “abc”
a.password = “def”
If a.getPassword() == “def”
    Return PASS
Else
    Return FAIL

Case 2:

User a;
a.userID = 1
a.username = “123”
a.password = “456”
If a.getPassword() == “456”
    Return PASS
Else
    Return FAIL

Case 3:

User a;
a.userID = 1
a.username = “#$@”
a.password = “-_-”
If a.getPassword() == “-_-”
    Return PASS
Else
    Return FAIL

Integration Tests:

Integration 1: Employee addCar(RentalCar, Location) : void

Case 1:

Employee a;
a.userID = 1;
a.username = “a”
a.password = “2”

RentalCar b;
b.carModel = “Civic”
b.carYear = 2018
b.carMake = “Honda”
b.isAvailable = True;
b.inGoodCondition = True;
b.renterName = “None”
b.carID = 3

Location c;
c.address = “La Mesa”;
c.rentalCars = []

a.addCar(b, c);

if(c.rentalCars[0] == b)
    Return PASS
Else
    Return FAIL

Integration 2: Employee findACar(Location, int) : RentedCar

Case 1:

Employee a;
a.userID = 1;
a.username = “a”
a.password = “2”

RentalCar b;
b.carModel = “Civic”
b.carYear = 2018
b.carMake = “Honda”
b.isAvailable = True;
b.inGoodCondition = True;
b.renterName = “None”
b.carID = 3

Location c;
c.address = “La Mesa”;
c.rentalCars = [b]

If a.findACar(c, b.carID) == 0
    Return PASS
Else
    Return FAIL

System Tests:

System 1: Interaction of Customer Accessing the Website or Mobile Application

Case 1:

When a customer opens up the website or mobile app, they will be prompted to share their location. If they do, nearby locations will be available for them to view. From there, they can choose whichever location they would like and directions to said location will be provided to them.

System 2: Interaction of Customer Paying for Their Rental Car

Case 1:

When a customer has selected a car ready for rental, they are directed to a screen that asks them to link a payment method. If the customer links the payment method, it will be stored in the system securely. The payment service will interact with bank and paypal servers so the payment is made to the car rental company.

Unit Tests Explanations:

Unit 1: getUsername() : string

Case 1:

The first case tests whether the getUsername() method will return the right username when it is alphanumeric. The input creates a User object with userID1, username “abc”, and password “2”. If a.getUsername() returns “abc”, the test passes. Fails otherwise.

Case 2:

The second case is testing if getUsername() will return the correct username when it is numeric. For the input, a User object is created with userID1, username “123” and password “2”. If a.getUsername() returns “123”, the test passes. Fails otherwise.

Case 3:

The third test case tests if getUsername() returns the correct username when it has special characters. For the input, User object is created with userID1, username “#$@”, and password “2”. If a.getusername() returns “#$@”, the test will pass. Fails otherwise.

Unit 2: getPassword() : string

Case 1:

The first case tests if getPassword() will return the correct password. For the input, a User object will be created with userID 1, username "abc", and password "def". If a.getPassword() returns “def” the test will pass. Fails otherwise.

Case 2:

For the second test case getPassword() returns the correct password when it’s numeric. A User object is created with userID 1, username "123", and password "456". If a.getPassword() returns “456”, the test passes. Fails otherwise.

Case 3:

The third case tests whether getPassword() returns the correct password when it contains special characters. A User object is created with userID 1, username "#$@", and password "-_-". If a.getPassword() returns “-_-” the test passes. Fails otherwise.

Integration Tests Explanations:

Integration 1: Employee addCar(RentalCar, Location) : void

Case 1:

In this integration test, we will be testing the addCar() operation of the Employee class. To begin, we created an Employee object and set its data members. We also did this for the RentalCar and Location objects. After the objects were created and set with their respective data, the Employee object called the addCar() funcion using the RentalCar and Location objects as its parameters. If the function worked correctly, then the first element in the Location object’s rentalCars array would be a RentalCar object that matches the one created by us earlier in the test. The first element in the Location object’s rentalCars array should be the RentalCar object we created, because that array was initially set to an empty array, and after the addCar function is called, the provided RentalCar object would be added into the rentalCar array. As such, it would become the first element in that array as that array was formerly an empty array.

Integration 2: Employee findACar(Location, int) : RentedCar

Case 1:

In this integration test, we will be testing the findACar() operation of the Employee class. To begin, we created an Employee object and set its data members. We also did this for the RentalCar and Location objects. After the objects are created and set with their respective data, the Employee object calls the findACar() function using the created Location object and the created RentalCar object’s carID data member as its parameters to search for an index in the rentalCars array of the created Location object where there is a RentalCar object with a carID data member that matches the parameter. For this case, if the returned index is a 0, then the function is working correctly. This is because the rentalCar array of the created Location object was initially set with a single element, which was the created RentalCar object. As such, its index position within the array would be a 0.

System Tests Explanations:

System 1: Interaction of Customer Accessing the Website or Mobile Application

Case 1:

For this test, it evaluates the interaction of when a customer accesses the application. They open the website or app and are prompted to a request asking if the customer's location can be accessed. If the customer grants access, nearby locations are displayed for them to choose. After selecting a location, a set of directions to the selected location is provided to them.

System 2: Interaction of Customer Paying for Their Rental Car

Case 1:

This test examines the payment process for when a customer is trying to rent a car. The system should guide the customer through linking a payment method securely and ensuring successful interaction with the bank and PayPal servers for payment processing.

Data Management Strategy:

Language: SQL

Changes To SWA Diagram:

- Break down Car Rental Database Into Other Databases
  - Customers
  - Employees
  - Rental Cars
  - Locations

Reason Behind Change:

The reason we changed from one large database to many smaller databases was because we believe that this change would make management of the data more user friendly and intuitive. This is because with a large database, all of our data is being crammed into one database so that means the table of our database will have many more fields to examine in one sitting. With many smaller databases, we make organization of the data table more clean and intuitive for users to examine. In addition, by linking each smaller database to each other, we can build relationships between the databases. These relationships help make the data more intuitive to understand.

An Example of Each Table for Each Database

- **Customers**

Customer ID	username	password	email	phone	name	rentedCar	Car ID	twoStep	Verified
1	Al	2468	al@gamil.com	123-456-7890	Al	false	0	true	true
2	Bonnie	abcd	bo@gmail.com	890-567-1234	Bonnie	true	1	true	true
3	Calvin	uwu	ca@gmail.com	109-238-4567	Calvin	false	0	false	true

- **Employees**

Employee ID	username	password
1	Abe	password
2	Bell	123456
3	Cat	=)

- **Rental Cars**

Car ID	Model	Year	Make	Available	Condition	Renter	Location ID
1	Camry	2024	Toyota	False	Good	Sam	1
2	Accord	2024	Honda	True	Good	None	1
3	Civic	2024	Honda	False	Bad	None	2

- **Locations**

Address	Car ID
La Mesa	1
La Mesa	2
San Diego	3

- **PaymentInfo**

Customer ID	Pay Type	Pay Info	Date	Cost
1	PayPal	al@gmail.com	11/11/11	200
2	Visa	1234-5678-9012-3456	02/22/22	180
3	Master Card	9876-5432-1098-7654	03/03/03	320

Strategy:
Our strategy is to use a total of 5 different databases to store our data. The employee database will have no association with the other 4 databases, and will be mainly used to store the accounts that will have permission to make changes within the system. For the customer database, this database will contain the accounts that will be renting cars from the company. These accounts will only be able to access the system, but not make any changes to the system. The customer database will interact with 2 other databases: the Rental Cars, and the PaymentInfo databases. For the Rental Car database, it will hold the data of the vehicles for rent by the company. This database will in addition to being associated with the customer database will also be associated with the locations database.The locations database stores the data of each respective branch of the company. Finally, the paymentInfo database will store the payment info of each respective customer.

Why SQL is better than No SQL
- SQL is easier to understand and organize than No SQL
- The structure of SQL is more organized
- Better organization for queries
- SQL can be followed better and consistently

Pros of having many smaller databases compared to a single bigger one

- Better simplicity
- More structured and organized
- Easier to spot problems
- Easier to maintain

Cons of having many smaller databases compared to a single bigger one

- Requires more resources to be allocated to create multiple databases
    - Requires more time
    - Additional hardware
        - Increases cost of system