Report #2

For

“Better Automobile Inventory Management”

CSCI441\_VA

Software Engineering

Fall 2019

<https://github.com/gculver/SoftwareEngineering_FinalProject>

Team:

Paul Whitely

James Cox

Grant Culver

September 9, 2019

Revision History:

|  |  |
| --- | --- |
| Version No. | Date of Revision |
| v.1 | 9/30/2019 |
|  |  |
| v.3 | 10/01/2019 |

**Contents**

Contents

1. Interaction Diagrams

2. Class Diagram and Interaction Diagram

2.1 Class Diagram………………………………………………………………………..?

2.2 Design Patterns..……………………………………………………………………..?

2.2 Data Types and Operations………………………………………………………….?

2.3 Tracability Matrix…………………………………………………………...............?

3. System Architecture and System Design

3.1 Architectural Style…………………………………………………………………..?

3.2 Indentifying Subsystems……………………………………………………………?

3.3 Mapping Subsystems to Hardware…………………………………………………?

3.4 Persistent Data Storage……………………………………………………………..?

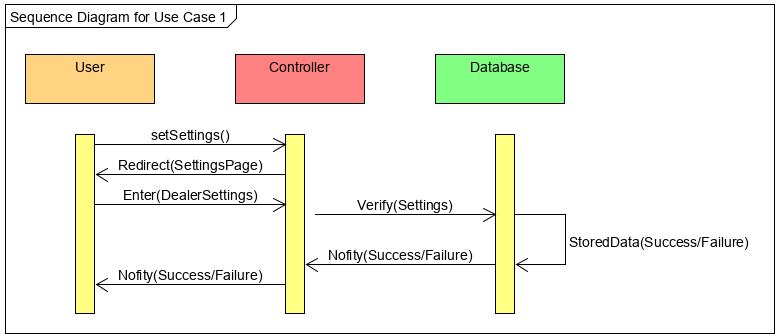
3.5 Global Control Flow………………………………………………………………..?

3.6 Hardware Requirements……………………………………………………………?

1. **Interaction Diagrams**

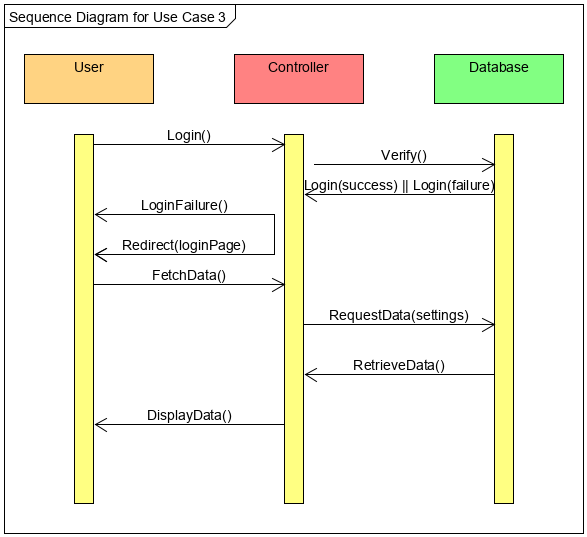
**1.1 Use Case 1: InvSettings**

We decided to assign the responsibility to set initial settings for the system to the controller, as mentioned by the **Expert Doer Principle** mentioned in textbook. This will allow the system to have a short communication chain between the related objects. The controller is the principle object and the secondary object in this situation would be the database. The database would be responsible to verify and store the settings that are received. In this instance, we believe it is necessary to use the publisher-subscriber design pattern to improve and implement this use case scenario. As related to this case, the user would be the subscriber and the system would be the publisher. Once the user subscribes valid input information, the publisher releases information of concern to the subscriber. I.E. A “success” message that notifies the user that the settings are set and acceptable. If the subscriber inputs invalid metrics, the publisher shows an error message showing invalid inputs.



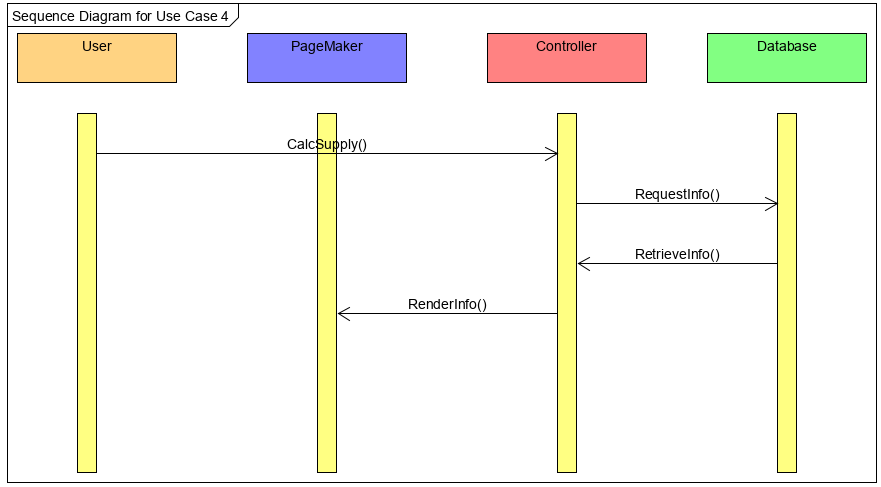
**1.2 Use Case 3: FetchData**

When a user attempts to fetch the data to calculate and display information from the user, the system attempts to verify the user authorization by querying the database for the correct account information. In this case, the databases main responsibility it to keep track of and report back to the user the relevant data. This data is retrieved by the system, which in-turn is responsible for passing information from the customer to the database and then relaying relevant information from the database to the customer based on the requests. For this interaction, we use the Publisher-Subscriber design pattern to improve this uses case’s design and functionality. When the user selects the desired data to be retrieved, they receive current inventory information that is able to correctly calculate the needed month’s supply and the inventory position. In this test case, the publisher gives the subscriber the pertinent information that the subscriber needs to make intelligent decisions.



**1.3 Use Case 4: CalcSupply**

Favoring the High Cohesion design principle the controller will be responsible for receiving information from the Database to calculate and display the inventory position and then pass the rendered page to the user.



**2. Class Diagram and Design Patterns**

**2.1 Class Diagram**



**2.2 Design Patterns**

The primary design pattern for this project will be the Model-View-Controller. The data model will be stored in a mySQL database and constraints and any necessary interface will be developed here. The Controller portion of the system will be written using PHP to interface with the database and external APIs. The system will present views of the information using HTML, Javascript and PHP.

**2.3 Data Types and Operations**

1. User

(a) Attributes

\* int user\_id: identification number of the user

\* string first\_name: customer’s first name

\* string last\_name: customer’s last name

\* string email: customer’s email address

\* string username: customer’s user name for system

\* string password: customer’s password for system

\* boolean isActive: customer’s status in the system

(b) Operations

\* signIn(): Allows the user to sign in to the system

2. FetchData

(a) Attributes

(b) Operations

\*getData(): fetches data from the database

3. InvSettings

(a) Attributes

\*int newCarMonthSupply: Overall new car month supply

\*int newTruckSupply: Month supply by truck lineup

\*int newCarSupply: Month supply by car lineup

\*int newCarStock: number of new vehicles in stock

\*int newTruckSupply: number of new trucks in inventory

\*int newCarSupply: number of new cars in inventory

(b) Operations

\*setNewCarMonthSupply(): set the desired monthly supply for new cars.

\*setNewTruckSupply(): set desired supply of new trucks.

\*setNewCarSupply(): set desired supply of new cars

\*getNewCarMonthSupply(): return setting of new car supply

\*getNewTruckSupply(): return setting of new truck supply

\*getNewCarSupply(): return setting for new car supply

\*storeData(): Stores settings in the database

4. CalcSupply

(a) Attributes

(b) Operations

\*calcSupply(): retrieves data from inventory settings and performs calculation

\*displayData(): Display calculated data to the user

5. Admin

(a) Attributes

\*string user\_name: username of the admin user

\*string password: password of the admin user

\*string name: name of the admin user

\*string email: email of the admin user

\*boolean isActive: status of the admin user in system

(b) Operations

\*createUser(): Create a new user

\*changePassword(): Change password of the user

\*changeStatus(): Change the attibute of the user isActive

\*deleteUser(): delete a user from the system

6. Database

(a) Attributes

\* DBConnection: Connects to the database

(b) Operations

\* connectDB(): Connects the database

7. Database Config

(a) Attributes

\* DB\_HOST: host of the database

\* DB\_NAME: name of database table

\* DB\_USER: user of the database

\* DB\_PASS: password to the database

(b) Operations

**2.4 Traceability Matrix**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Classes | Domain Concepts | | | | | | | | |
| Database |  |  |  |  |  |  |  |  |
| User |  |  |  |  |  |  |  |  |  |
| FetchData |  |  |  |  |  |  |  |  |  |
| InvSettings |  |  |  |  |  |  |  |  |  |
| CalcSupply |  |  |  |  |  |  |  |  |  |
| Admin |  |  |  |  |  |  |  |  |  |
| Database |  |  |  |  |  |  |  |  |  |
| DB\_Config |  |  |  |  |  |  |  |  |  |

asas