Report #3

For

“Better Automobile Inventory Management”

CSCI441\_VA

Software Engineering

Fall 2019

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

Team:

Paul Whitely

James Cox

Grant Culver

November 5, 2019

**Contents**

1 Customer Statement of Requirements

2 Glossary of Terms

3 System Requirements

4 Functional Requirements

4.1 Implemented Use Cases

4.2 Casual Description of Unimplemented Use Cases

4.3 System Sequence Diagrams

4.4 Traceability Matrix

5 Effort Estimation Using Use Case Points

5.1 Actor Classification

5.2 Use Case Classification

5.3 Technical Complexity Factors (TCFs)

6 Domain Analysis

6.1 Traceability Matrix

7 Interaction Diagrams

8 Class Diagram and Interface Specification

8.1 Design Patterns

8.2 OCL Contracts

9 System Architecture and System Design

10 Algorithm and Data Structures

11 User Interface Design and Implementation

12 Design of Tests

13 History of Work, Current Status, and Future Work

14 References

1. **Customer Statement of Requirements**

**1.1 Problem Statement**

Throughout the history of civilization and business, there has been a continuous trend towards increased sophistication and efficiency. Sometimes these advancements are small and seemingly insignificant and other times they are drastic and life altering. One can look back at the Gutenberg Press and see that it was a major advancement in education and society. The industrial revolution changed the way products were manufactured and produced. It gave way to the automobile which fundamentally changed society and business by effectively enlarging the geographic area that businesses could compete in. Today, businesses face another revolution and that is the technological revolution that is completely altering the landscape for society and business. It has given way to self-driving cars, space shuttles that can save enormous sums of money by reusing booster rockets and in the retail business, given rise to the “Goliath” of Amazon. Amazon is fundamentally altering the rules and the playing field that small business must compete in. It does no good to debate the equality or the goodness or badness to society of such a mammoth company. Amazon will force, through survival of the fittest, businesses to adapt and modify the way they engage and profit in today's business climate. It is both exciting and perplexing to be a part of. Fortunately, the same technology that has enabled amazon to flourish and prosper will also enable small businesses to perhaps gain an advantage and compete toe-to-toe with a seemingly unbeatable “Goliath”. This has given rise to companies such as salesforce that allow small business to compete on cost with many larger competitors. Open source software has driven down the cost for many businesses. Our company reduced our CRM expense by over $150,000 per year due to these developments. Our proposed solution plans to start the process by enabling our company to economically develop a new car inventory management system with the goal of maximizing shareholder profit by reducing expense and providing relevant, convenient and up-to date information.

The goal of a new car inventory management system is to understand quickly and easily where in relation to a model inventory our current inventory stands. A model inventory is based 3 variables. First, the dealership must be informed as to what the past 3 months sales history looks like regarding total new vehicles sold by brand and then broken down by model. Industry standard is to have a four month’s supply of new car inventory. Second, the dealership must realize what the current inventory levels are by brand and by model. Third, the dealership must be informed as-to whether it is short or long in inventory by model and brand. Currently all of this information is scattered over 3 or more systems and leads to frustration and indigestion. Our customer centric solution is to consolidate these 3 systems into 1 system that will ultimately save the customer money and time.

The monetary savings will largely be due to three main factors. First, the dealership will no longer be stocking inventory it doesn’t need, or that it is too long in. When considering 10,000,000 dollars in inventory at 4.5% interest, stocking the right inventory is not a matter of just increased profitability but ultimately of long-term survivability. Second, savings will be increased by the lessening of time it takes to tackle implementing a model inventory. There will no longer be a need for logging into 3 different systems to obtain information. This will allow user to focus on other business-related tasks and ultimately be more efficient. Thirdly, this will save money due to the unnecessary expense of paying for 3 complimentary but uniquely different software solutions. With the combination of all three features of creating a model inventory software solution, additional or unneeded software can be cancelled.

Additional features of the software solution will include authentication to access the system. System will allow additional users to register or sign-up according to dealership policy. The system will require dealership setting up the recommended months supply with a degree of granularity for specifying months supply not only for vehicle make but also by model. The user interface will be user friendly and will focus heavily on keeping system user friendly with clutter kept to a minimum. The system will utilize a combination of API feeds and database feeds to gather required information for system to calculate model inventory.

1. **Glossary of Terms**

To provide better understanding of the contents of this report, listed below are important terms of our proposed software solution:

**Month’s Supply** – Units in stock (at month end) divided by Sales history (in months).

**Model Inventory** – Guideline for stocking new car inventory that takes into account past sales history, user defined month’s supply and the output is recommended number of units in stock.

**Long Inventory** – Inventory in-stock that is above model inventory suggestion.

**Short Inventory** – Inventory in-stock that is less than model inventory suggestion.

**Make** – Manufacturer of vehicle. Ex. Jeep, Dodge, Chevrolet, Toyota, etc.

**Model** – Particular model of vehicle made by a respective manufacturer.

Ex. Wrangler, Silverado, 1500, 4-Runner, Land Cruiser, etc.

**Inventory Manager** – (AKA New Car Inventory Manager) – is principle person concerned with managing state of inventory and forecasting sales in the coming months.

3 System Requirements

3.1 Enumerated Functional Requirements

|  |  |  |
| --- | --- | --- |
| Identifier | Priority  Weight (Low 1 - 5 High) | Requirement Description |
| REQ-1 | **3** | System shall support user login and authentication |
| REQ-2 | 1 | System shall support registering new users. |
| REQ-3 | 4 | System shall support users to define month supply of inventory by both make and model |
| REQ-4 | 5 | System shall pull in-stock inventory from dealership CRM |
| REQ-5 | 2 | System shall calculate month’s supply of inventory |
| REQ-6 | 4 | System shall compute long or short position of inventory |

3.2 Enumerated Non-Functional Requirements

A model called FURPS+ will be used here to qualify software attributes, which stand for functionality, usability, reliability, performance, supportability and the + stands for other possible attributes needed. We will be focusing on the non-functional requirements which cover FURPS+.

Functionality: Our tool should satisfy the needs of the customer such that they no longer have the need to subscribe to multiple inventory tools for their dealership. The software should use an API to gather the information that it needs to calculate model inventory.

Usability: The system should be easy to use and simple to understand. The software to consistently give accurate inventory model information so that the customer can rely on it when making decisions regarding their inventory.

Reliability: The system should give accurate and reliable results regarding the needs of the dealership and the inventory on hand.

Performance: The system to be should have high performance. The process of signing in and receiving data should take no more than 5 minutes.

Supportability: The system is benign and user friendly. The learning curve will be modest and user documentation will introduce designed program usage and will contain a software demonstration video or text. Support will all be available via email to handle software bugs and imperfection.

4 Functional Requirements

4.1 Automotive Professionals and Administrative Staff

This software is principally designed for senior level, mid-level automotive managers and administrative staff (i.e. accounting department).

In the current business environment and as business evolves to a more data driven approach to inventory management, people are going to make decisions based on previous data, and therefore is a defining characteristic of this software solution. Hence, in this software, all agents make decisions according to previous sales results. The principle stakeholders of this system are mentioned below.

The principle stakeholder of this software proposal is the new car inventory manager or inventory manager. This persons job exists to minimize carrying costs associated with stocking inventory and maximizing dealership profit by stocking the right make and models based off prior selling history and future market conditions.

The General Manager or Owner of the Automotive dealership is another stakeholder. Their interaction with the system to be would be more monitoring and ensuring utilization of the software system.

The controller would ensure that billing of the software solution is completed within agreed upon terms and would likely use the portal to ensure correct billing and payment processing.

4.2 Actors and Goals

|  |  |  |
| --- | --- | --- |
| **Actor** | **Actor’s Goal** | **Use Case Name** |
| User | Fetch Database Records | FetchData(UC – 3) |
| Admin | Set up user | UserAdmin (UC – 2) |
| User | Set Inventory Settings | InvSettings (UC – 1) |
| User | Calculate Month’s Supply | CalcSupply (UC – 4) |
| User | Model Detail | ModelDetail (UC-5) \*new use case |
| Admin | Pay Bill | BillPay(UC – 5) \*unimplemented |

4.3 Use Cases

|  |  |  |
| --- | --- | --- |
| **UC – 1** | | **InvSettings** |
| Related requirements | |  |
| Initiating actor | | User |
| Actor’s goal | | Set initial settings for evaluation |
| Participating actors | | System, User |
| Preconditions | | User is active and initial screen is showing |
| Post conditions | | Initial settings are set and evaluation is ready to start |
| **Flow of events for main success scenario** | | |
|  | User (a) selects the menu item “Settings” (b) types in Month’s supply | |
|  | User (a) clicks save | |
|  | System verifies values and stores them in database | |
| **Flow of events for alternate scenario** | | |
|  | User (a) submits invalid Month’s Supply | |
|  | System prompts user for valid Month’s supply | |
|  | User (a) enters valid Month’s supply | |
|  | System verifies values and stores them in database | |

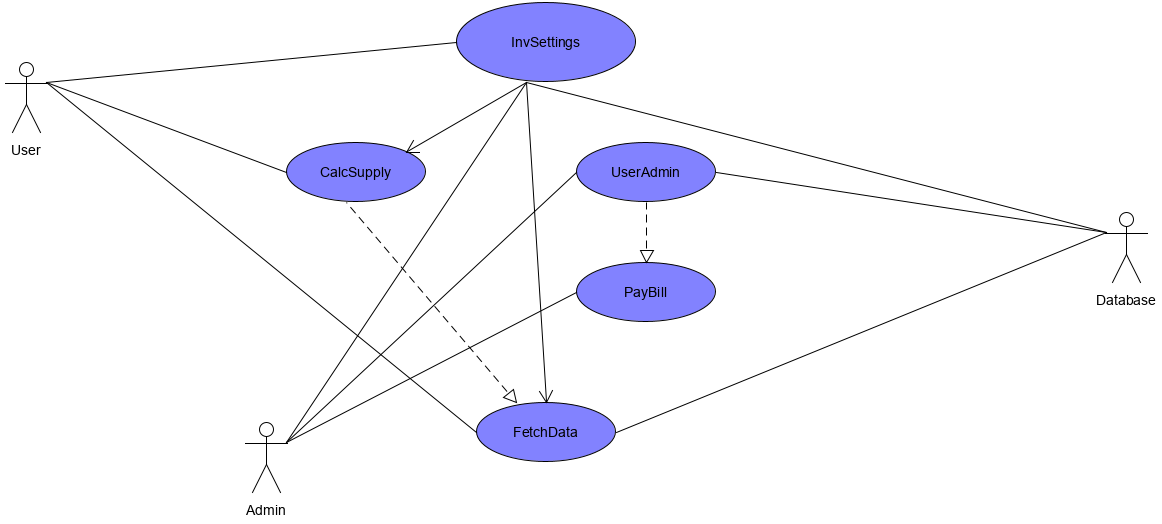
|  |  |  |
| --- | --- | --- |
| UC – 2 | | UserAdmin |
| Related requirements | |  |
| Initiating actor | | Admin |
| Actor’s Goal | | Set up an authorized user account |
| Participating actor | | Database, User |
| Preconditions | |  |
| Post condidtions | | User account is active |
| **Flow of events for main success scenario** | | |
|  | Admin user clicks “Admin” icon on home page | |
|  | Admin user submits required information | |
|  | System stores user in database and shows “Success” to Admin user | |
| **Flow of events for alternate scenario** | | |
|  | Admin user submits user info | |
|  | System notifies Admin user is already active | |

|  |  |  |
| --- | --- | --- |
| UC – 3 | | FetchData |
| Related requirements | |  |
| Initiating actor | | User |
| Actor’s goal | | Pull sales history from database to allow comparison. |
| Participating actor’s | | Database |
| Preconditions | | System is connected to database |
| Post conditions | | Data is pulled from database |
| **Flow of events for main success scenario** | | |
| C | User clicks button to initiate download of data | |
|  | Software solution pulls data from database | |
|  | Software sends success message to users browser | |
| **Flow of events for alternate scenario** | | |
|  | User clicks button to initiate download of data | |
|  | System notifies user connection cannot be established | |

|  |  |  |
| --- | --- | --- |
| UC – 4 | | CalcSupply |
| Related requirements | |  |
| Initiating actor | | User |
| Actor’s goal | | To calculate month’s supply of inventory based off previous sales data |
| Participating actor | | Database, User |
| Preconditions | | Month supply calculation and settings entered correctly |
| Postconditions | | Data is displayed to user |
| Flow of events for main success scenario | | |
|  | User clicks calculate supply button | |
|  | Software calculates Months supply and displays to user | |
| Flow of events for alternate scenario | | |
|  | User clicks calculate supply button | |
|  | Settings are incorrect or not defined software sends error message to interface | |

|  |  |  |
| --- | --- | --- |
| UC - 5 | | ModelDetail |
| Related requirements | |  |
| Initiating actor | | User |
| Actor’s goal | | Detail by inventory model. Goal is to show user time to turn by trim package. |
| Participating actor | | Vendor System |
| Preconditions | | User is logged in and inventory is displayed to user. |
| Postconditions | | User knows which trim packages are best sellers. |
| Flow of events for main success scenario | | |
|  | | User logs into system and inventory is displayed. |
|  | | User clicks model link on inventory display page. |
|  | | System display inventory details based on model. |
| Flow of events for alternate scenario | | |
|  | User clicks link for no inventory data | |
|  | System produces flash or alert message stating there is not inventory data to display | |

4.3.1 Use Case Diagram

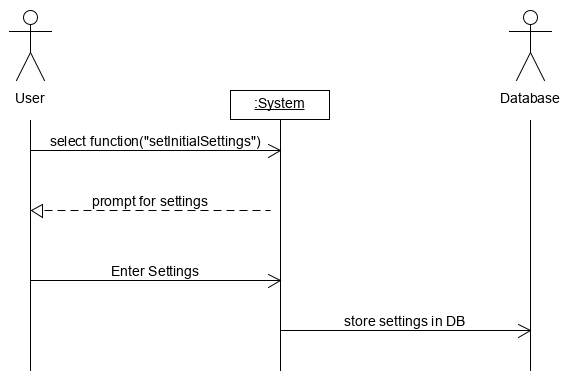


4.3.2 Traceability Matrix

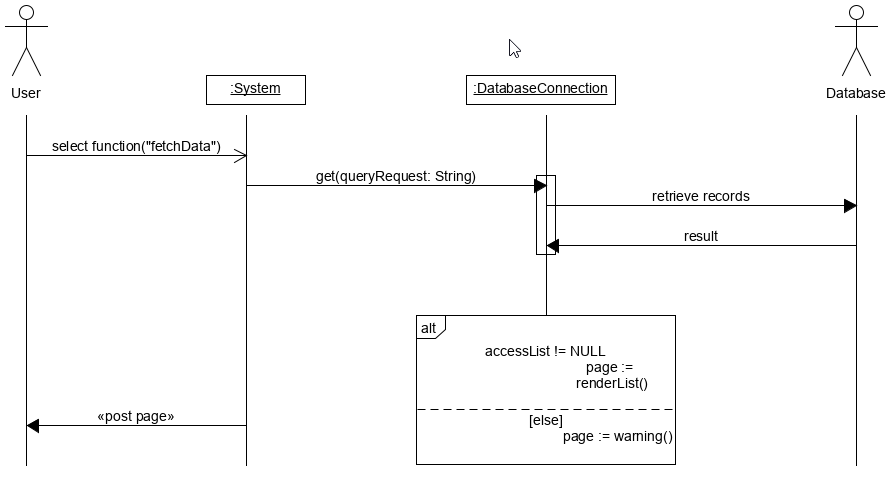
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Req’t | PW | UC1 | UC2 | UC3 | UC4 | UC5 |
| REQ1 | 3 | X |  |  |  | X |
| REQ2 | 1 | X |  |  |  |  |
| REQ3 | 4 | X |  | X |  | X |
| REQ4 | 5 |  |  | X |  |  |
| REQ5 | 2 |  |  |  | X |  |
| REQ6 | 4 |  |  |  | X |  |
| MAX PW |  |  |  |  |  |  |
| TOTAL PW |  |  |  |  |  |  |

4.3.3 System Sequence Diagrams

UC-1: setInvSettings



UC-2: FetchData



5. Effort Estimation using Use Case Points

5.1 Actor Classification

UAW : 3(2) + 2(2) + 1(1) = 11. The unadjusted actor weights are determined by the complexity and interaction with the user interface.

|  |  |  |  |
| --- | --- | --- | --- |
| **Actor** | **Description** | **Complexity** | **Weight** |
| User | Users will use graphical web interface for site. | Complex | 3 |
| Admin | Admin will setup users and manage site. | Complex | 3 |
| Code | Set Inventory Values | Simple | 2 |
| Code | Calculate Month’s Supply | Simple | 1 |
| Admin | Pay Bill for services | Average | 2 |

5.2 Use Case Classification

UUCP: 3(5) + 1(10) + 1(15) = 40

|  |  |  |  |
| --- | --- | --- | --- |
| **Use Case** | **Description** | **Complexity / Weight** |  |
| FetchData(UC – 3) | Fetch data from database. 1 actor. | Simple | 5 |
| UserAdmin (UC – 2) | Admin user management. 1 actor. | Complex | 15 |
| InvSettings (UC – 1) | Setting inventory parameters. 1 actor | Simple | 5 |
| CalcSupply (UC – 4) | Calculation logic.  1 actor. | Simple | 5 |
| BillPay(UC – 5) | Paying bill for services. 1 actor. | Average | 10 |

5.3 Technical Complexity Factors (TCFs)

TCF Total : 32

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Technical Factor** | **Description** | **Weight** | **Perceived Complexity** | **Factor**  **(W\*PC)** |
| T1 | Web and Database communication. | 2 | 3 | 6 |
| T2 | User perceived performance of application | 1 | 4 | 4 |
| T3 | Calculation of Auto Supply | 2 | 3 | 6 |
| T4 | Site Administration | 2 | 2 | 4 |
| T5 | Inventory management | 1 | 2 | 2 |
| T6 | Site installation and setup | 1 | 3 | 3 |
| T7 | Importing historical data | 2 | 2 | 4 |
| T8 | User training | 1 | 0 | 0 |
| T9 | Analyze trends from historical data | 1 | 3 | 3 |

**6. Domain Analysis**

6.1 Domain Model

For the purpose of building a domain model, it is a necessity to review use cases and system requirements to identify relations between use cases and the bearers of responsibility that are required to implement and achieve the goals of the use cases. These concepts are paramount to achieving a successful model that will help implement our software solution. We shall then expound on the concepts by identifying attributes and associations.

6.1.2 Concept Definitions

As mentioned in the text, we feel the necessity to have a responsibility doer as the primary concept. This doer, from here on noted as controller, has the responsibility to be the “front line” concept that will initialize action and delegate responsibility to the rest of the concepts.

6.1.3 Boundary Concepts

|  |  |  |
| --- | --- | --- |
| Responsibility Description | Type | Concept Name |
| Coordinate actions of all concepts associated with a use case, a logical group of use cases, or the entire system and delegate the work to other concepts. | D | Controller |
| Container for user authentication. Login, password, and active. | K | Database |
| Container for administrative authentication. Setting up and deleting a user. | K | Database |
| Container for inventory settings | K | Databse |
| Log Interactions w/ system in perisistent storage | K | Database |
| Fetch Inventory from database | D | FetchInv |
| Calculate Inventory Supply | D | Calculator |
| Display Inventory position. | N | Visual Interface |
| Allow users to login | N | Visual Interface |
| Allow users to set settings | N | Visual Interface |
| Prompt User to get inventory | N | Visual Interface |

The property of these concepts includes types, notably, the K type and D type as shown on forthcoming domain model graph. The “smiley” or “document” symbol tagged on each concept. From the textbook, author compares K and D to things and workers. Workers are assigned mainly doing responsibilities and things are assigned knowing responsibilities. The following is the concept diagram divided by K or D with K being symbolized by document and D symbolized by smile.

6.1.4 Association Definitions

Associations with different concepts are listed below. These arrows indicate the relationship between each concept and mainly for conveying information and saving related information.

|  |  |  |
| --- | --- | --- |
| Concept Pair | Association Description | Association Name |
| Controller Page Maker | Controller passes requests to page maker and receives back pages for displaying | Conveys requests |
| Page Maker DB Connection | DB Connection passes retrieved data | Provides Data |
| Controller DB Connection | Controller passes search requests to DB Connection | Conveys requests |
| Page Maker Visual Interface | Page Maker prepares the interface page | Prepares |
| User DB Connection | Agent stores settings in DB Connection | Settings Save |
|  |  |  |
|  |  |  |

6.1.5 Attribute Definitions

|  |  |  |
| --- | --- | --- |
| **Concept** | **Attributes** | **Attribute Description** |
| Settings | Day Supply | Used to determine stocking position of inventory |
| Settings | User Credentials | Used to determine if user has access to the system |
| Controller | Received data | Information received from the database |
|  |  |  |
|  |  |  |
|  |  |  |

6.1.6 Traceability Matrix

|  |  |  |
| --- | --- | --- |
| **Use Case** | **PW** | **Domain Concepts** |
| **Controller Settings Fetcher Bill\_Pay Calc\_Supply** | | |
| UC1 |  | X  X  X X  X X  X X |
| UC2 |  |
| UC3 |  |
| UC4 |  |
| UC5 |  |

6.2 System Operation Contracts

UC-1: InvSettings

* Preconditions
  + User is an active user
  + User has logged into system
  + User has appropriate authorization to change system settings
* Postconditions
  + User has set appropriate settings for model inventory
  + User is shown a “success” message

UC-2: UserAdmin

* Preconditions
  + System administrator is logged into system
* Postconditions
  + System administrator sets up an authenticated & authorized user
  + User will have a ID, Password and email associated with account
  + Settings are stored in database

UC-3: FetchData

* Preconditions
  + User is logged in with correct authorization
* Postconditions
  + System displays fetched data from database connection
  + User sees “success” acknowledgment message

UC-4: CalcSupply

* Preconditions
  + User has set acceptable parameters for months supply.
  + Or system default is in place.
* Postconditions
  + System calculates and display month supply data
  + System calculates and displays inventory position as either short or long.

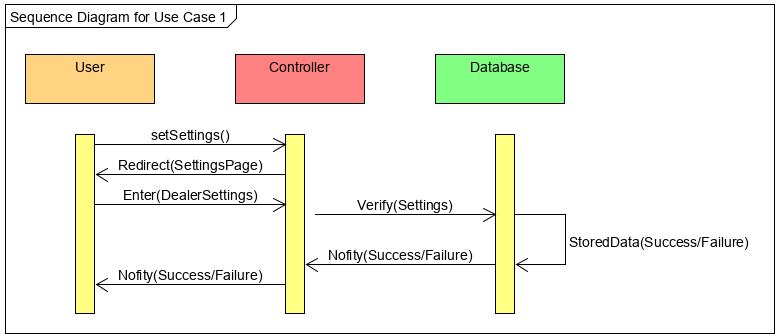
UC-5: ModelDetail

* Preconditions
  + User is logged into system
  + Data is fetched from database
* Postconditions
  + System displays model detail to user

7. Interaction Diagrams

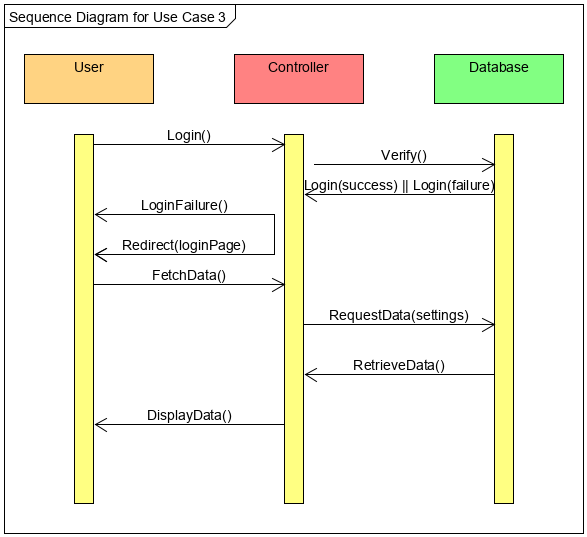
**Use Case 1: InvSettings**

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.



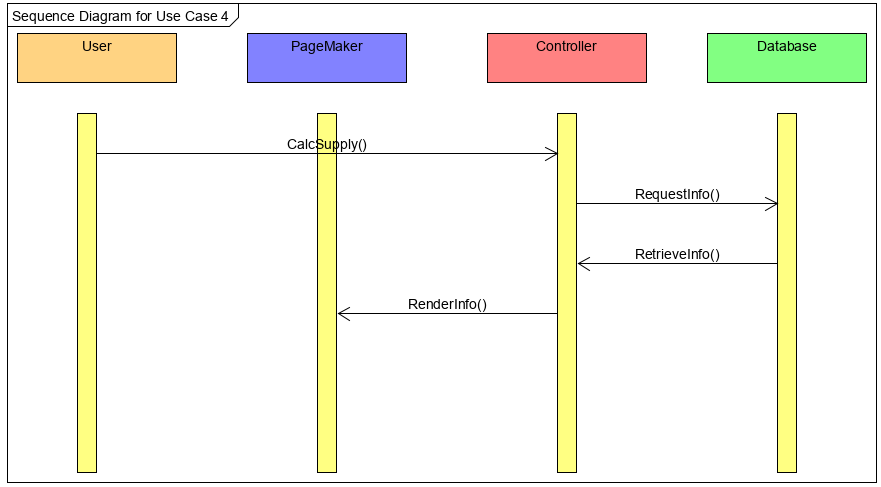
**Use Case 3: FetchData**

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.



**7.2 UC 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.



**8 Class Diagram and Interface Specification**



**8.1 Design Patterns**

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

In a more abstract terms, hence utilizing the MVC framework, for the majority of this application we will be utilizing the Publisher-Subscriber Design Pattern. When the subscriber executes certain actions, e.g. fetching inventory from the database, the publisher will send the subscriber the relevant information via database connection and represented graphically in GUI. Utilizing the publisher-subscribed design pattern will reduce coupling within our application and will ensure that when updates or modifications are made a complete reconfiguration of the system will not be needed.

**8.2 Object Constraint Language (OCL)**

The classes will have some preconditions to them. The Automobiles will need to be valid makes models and years to be loaded into the database. There must be positive integer values for actual quantities, although placeholder values (such as zero or negative one) might be used to indicate certain conditions to be determined. Registered user information will need to be validated before it can be submitted to the database.

9 System Architecture and System Design

In terms of structure for the application logic modules, the inventory system will utilize a component-based design style. Each task that must be completed within the system is performed by a module dedicated to that task (the database class handles the connection to the database, the config system handles the database configuration settings, the inventory settings module handles specifically the handling of the settings and is independent of inventory calculations).

As for memory and data sharing, the system uses a database-centric architecture. All information relating to user settings, admin settings, and inventory settings are saved in databases which are accessible to other components or modules in the system. This database design allows the various components of the system to access, view, and edit the same data, allowing for easing communication between components in many cases.

The direct communication between components in the system is also heavily based on an event-driven architecture. Each primary function in the system must happen in a predefined step during a normal use case. For example, the inventory settings must be addressed and completed before calculation and display of inventory position to the end user. Each step in the system’s process is triggered by the completion or progress of a previous step, which is the main philosophy behind an event-driven design.

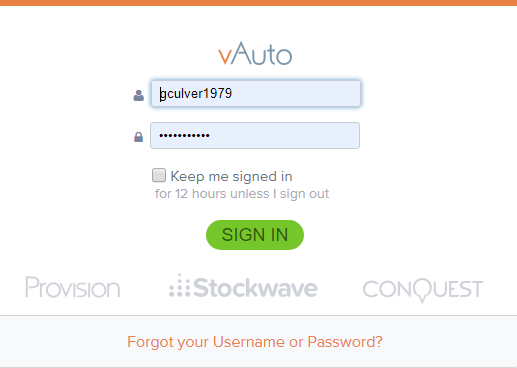
10 Algorithms and Data Structures

The Better Automobile Inventory Management will collect information from various government sources regarding automobile sales. Averages for a timeframe will be calculated and compared to current and projected inventory levels. Statistical outliers will be noted so that human judgement can be used to make correct decisions.

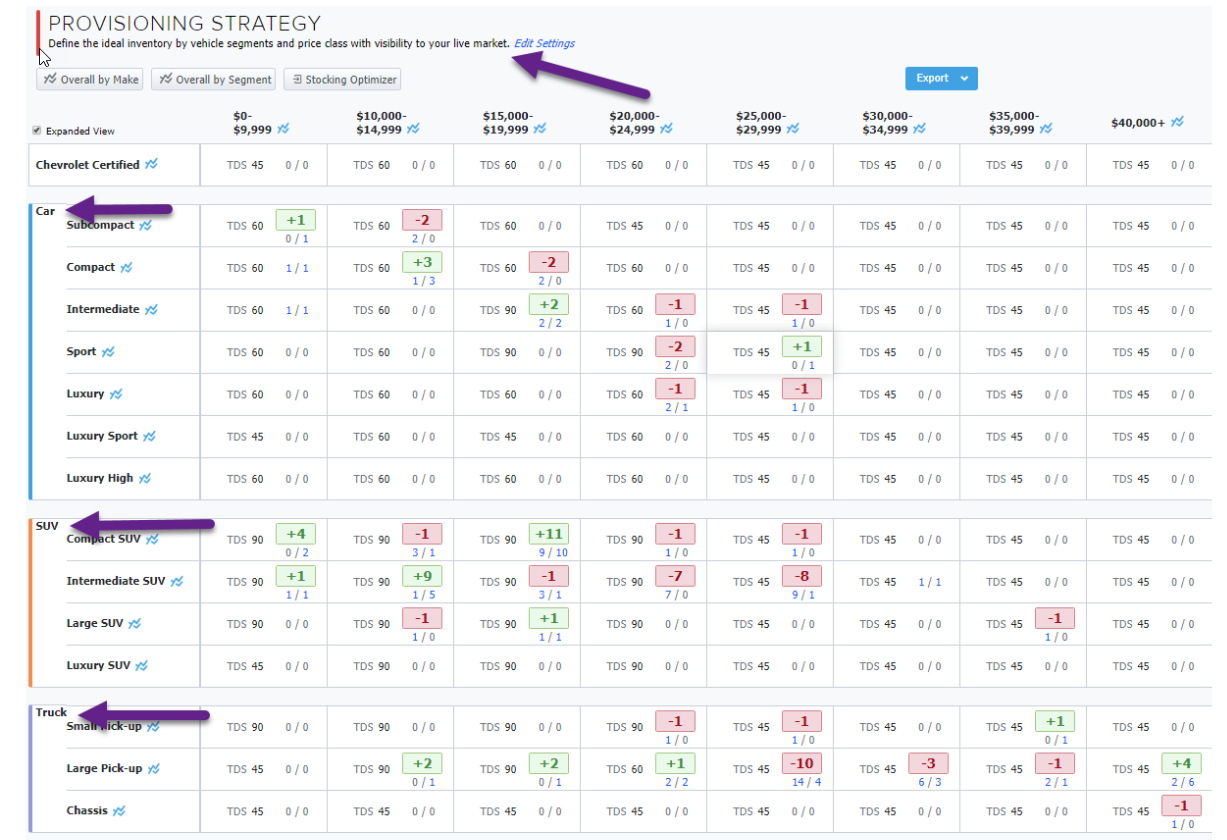
Data structures will all be stored within the database. Data schemas will be designed to ensure that only valid data is stored. Application data entry will be validated before attempting to submit to the database.

11 User Interface Design and Implementation

* 1. Sign-In



* 1. Settings



* 1. calcSupply



12 Design of Tests

1. Login:
   1. A registered user can input their username and password to log into their account.
   2. The webform prevents a login using an invalid username and valid password.
   3. The webform prevents a login using a valid username and invalid password.
2. Registration:
   1. A new user can input their information and register for an account. All the data must be collected via a web form.
   2. Information given upon registration will be saved in the database.
   3. Test for invalid entries when registering a new user.
   4. Test for blank fields when registering a new user.
   5. Testing can be done via querying of database of the new user.
3. Settings Data:
   1. Settings data can be input into user form and will be stored in the database.
   2. Create tests for each operation that they function as expected.
      1. createUser()
      2. changePassword()
      3. changeStatus()
      4. deleteUser()
   3. Testing can be done against the database by querying for information.
4. Calculate Supply:
   1. A logged in user can query information from the database and system will display correct information. Testing can be done with stub driver.

**13 History of Work, Current Status, and Future Work**

**13.1 Key Accomplishments**

* Implemented the mongo, express, node stack to meet project requirements.
* Collaborated in a meaningful and timely fashion to meet project deadlines.
* Used various problem solving methods to improve and get code to working correctly. E.g. github, udemy and work colleagues.

**13.2 Future work on project**

While current software solution is working and usable there are numerous future enhancements that could not only improve the performance but also make the solution a commercially viable product. Depending on time constraints we may or may not implement database API’s in current solution to make it 100% free of any user involvement on the setup. Also, there exists an opportunity to pull data from an external API to show and track the dealership’s market share. Currently, and as previously mentioned, there is no steadfast metric for measuring performance that is currently widely used or accepted. It is our believe that one such metric is market share and this data could be pulled into the system via an API from Polk Data Research. This would show new car registrations by zip code and the system would be able to filter and display a dealerships market share by these zip codes and the new car registrations in each respective zip code. Also, a web scrapper could be utilized to compare inventory versus a competitor to show inventory differences and opportunities. In addition, data could be pulled and collected that would show how long it took to sell a model and the gross per model. With a good algorithm, this could be enhanced to give a quality picture of what a model inventory would look like.

**14 References**

The application will be using Node.js, Express.js and MongoDB to allow rapid development and ease of use.

<https://nodejs.org/>

<https://expressjs.com/>

<https://www.mongodb.com/>

We will be using wrike.com as a project management tool to allow us to keep on track and complete the project on time.

<https://www.wrike.com/>

We will be using github as a version control software so that we can work together as a team in the same repositories and have a version control to professionally manage our development.

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