Heavy Machinery Rental System

CS4125 Team-Based Project Report

Team Members

Conor Canton - 16164571
Eoin Flynn - 15179818
Murdo Mackenzie - 16152522
Panos Brennan Andreou - 16158687
Vainqueur Kayombo - 17199387

GitHub Repo: https://github.com/VainqueurK/lender

Commit ID: 33f6bdec7040cdc0289e02ca971c6f76003f3158

https://github.com/VainqueurK/lender/releases/tag/1.2

Table of Contents

Business Scenario

Our business application revolves around the construction industry and enabling easy access to large corporations to hire out specialised machinery which they might only need for a short period or more common machinery which could be loaned out for a longer duration based on the project's length. We will offer a multitude of different machinery options. Each machine will have a max number of working weeks it can be rented for due to certain demand and popularity of this machine.

Our target market is large scale construction companies. Delivery will be available with each rental along with collection. Each machinery must be filled back to the top with fuel like when it was received and if this is not the case it is possible for a fine to be added onto the account of the business. All our machines will be fitted with telemetry equipment which will not only help keep the vehicle safe if lost or stolen but also allow us to accurately track the machines conditions and working hours to ensure that the relevant maintenance can be carried out on a machine when it has been returned from a rental

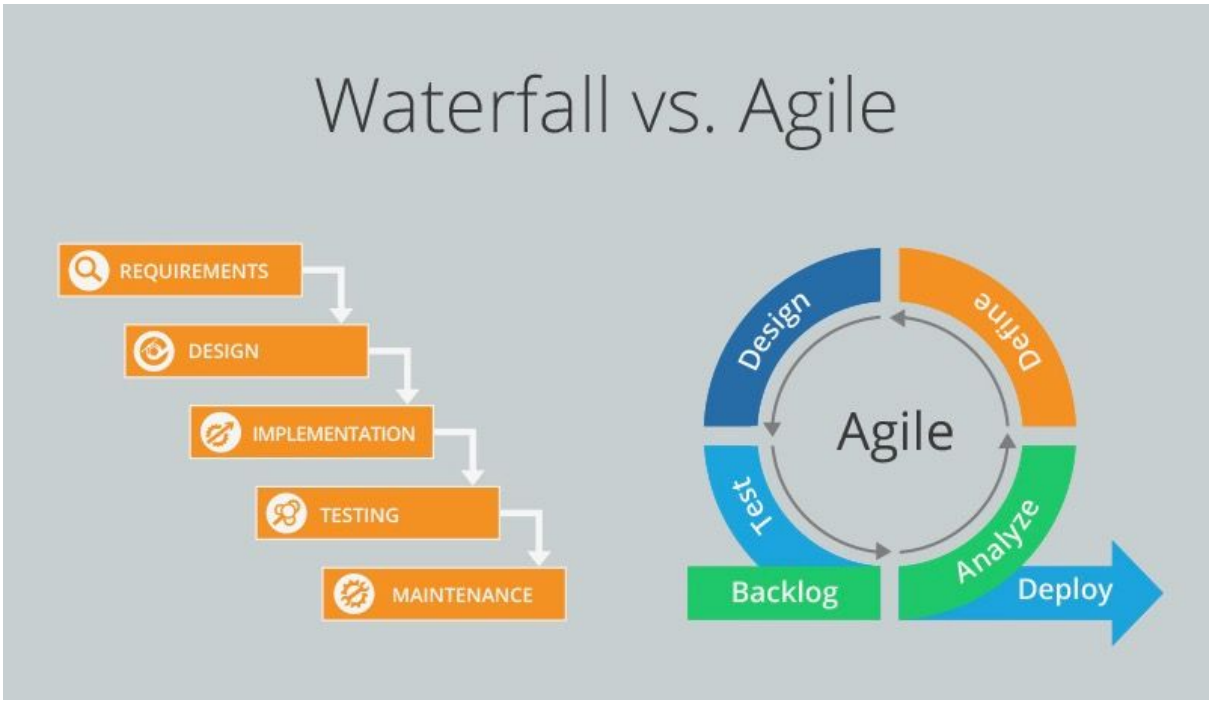
This machinery rental management tool will allow for companies who need a specific machine to easily check for its availability, price, specifications and make a rental for a specific time period.

Rental is a sustainable sharing economy business in several aspects and a more circular economy is what's required to make sure a sustainable future. By sharing tools and equipment we tend to contribute to carbon reduction.
Sharing machinery between multiple clients we cut down on the amount of machinery that needs to be manufactured/purchased.

Software Lifecycle

The first plan we came up with was the most logical one, a linear approach where things are done out in sequence after separating them into milestones and heading, similar to a waterfall model. For example, after deriving the requirements and starting the design we would then go into implementation, but nothing is done in parallel and the implementation is bulk verified after completing an initial set of implementations. We concluded that this model is not as lightweight or mutable as we would like and required robust specifications for the requirements and design where planning needed to be near perfect for it to function effectively.

After some research, we decided to go with the Agile approach where we could work in sprints while implementing a continuous model for requirements and planning, where we could change things after each successive sprint. This allows for change to occur quickly and efficiently. Agile also focuses and promotes people to work together; it allows for solutions to evolve from collaboration.



Project Plan

Deliverable	Description	Responsibility	Week
Narrative Description	Describe the business scenario	Conor Canton	6
Software Lifecycle	Describe and justify the chosen software lifecycle	Vainqueur Kayombo	6
Established Roles	Specify the roles that people will have to fill out in order to complete the project	Vainqueur Kayombo	6
Requirements	● Functional Requirements ● Database ● Use Case Diagrams ● Use Case Description ● Non Functional Requirements ● Plans for Quality Attributes ● User Interface screenshots	Conor Canton Panos Brennan Andreou Conor Canton Conor Canton Panos/Murdo Murdo Mackenzie Panos Brennan Andreou	7
System Architecture	● Package Diagram ● Architectural Decision Justification	Eoin Flynn Eoin Flynn	8
Analysis Sketches	● Candidate Classes ● Analysis Class Diagram ● Collaboration Diagram ● Communication Diagrams ● Entity Relationship Diagram	Eoin Flynn Eoin Flynn Conor Canton Murdo Mackenzie	9
Code	Code Implementation	Everyone	12
Design	● Architectural Diagram ● Class Diagram ● State Chart	Eoin Flynn / Panos Murdo Mackenzie	12
Critique	● Overview ● Design ● Implementation	Panos	12
References	List of Sources	Murdo Mackenzie	12

Project Roles

Roles	**Description**	**Team Member**
Project Manager	Tracks progress, sets up meetings and creates project plans	Vainqueur Kayombo
Documentation Manager	In charge of sourcing relevant	Murdo
Architect	Designs system architecture	Eoin
Requirements Engineer	Derives requirements	Conor
Systems Analyst	Creates class models	Eoin/Vainqueur/Panos
Designer	Responsible for recovering design time blueprints	Panos/Murdo
Dev Operators	Responsible for development	Everyone
Technical Lead	Leads the main implementation	Panos
Tester	Coding automated tests	Vainqueur/Eoin/Murdo
Programmers	Responsible for programming	Vainqueur/Conor

Requirements

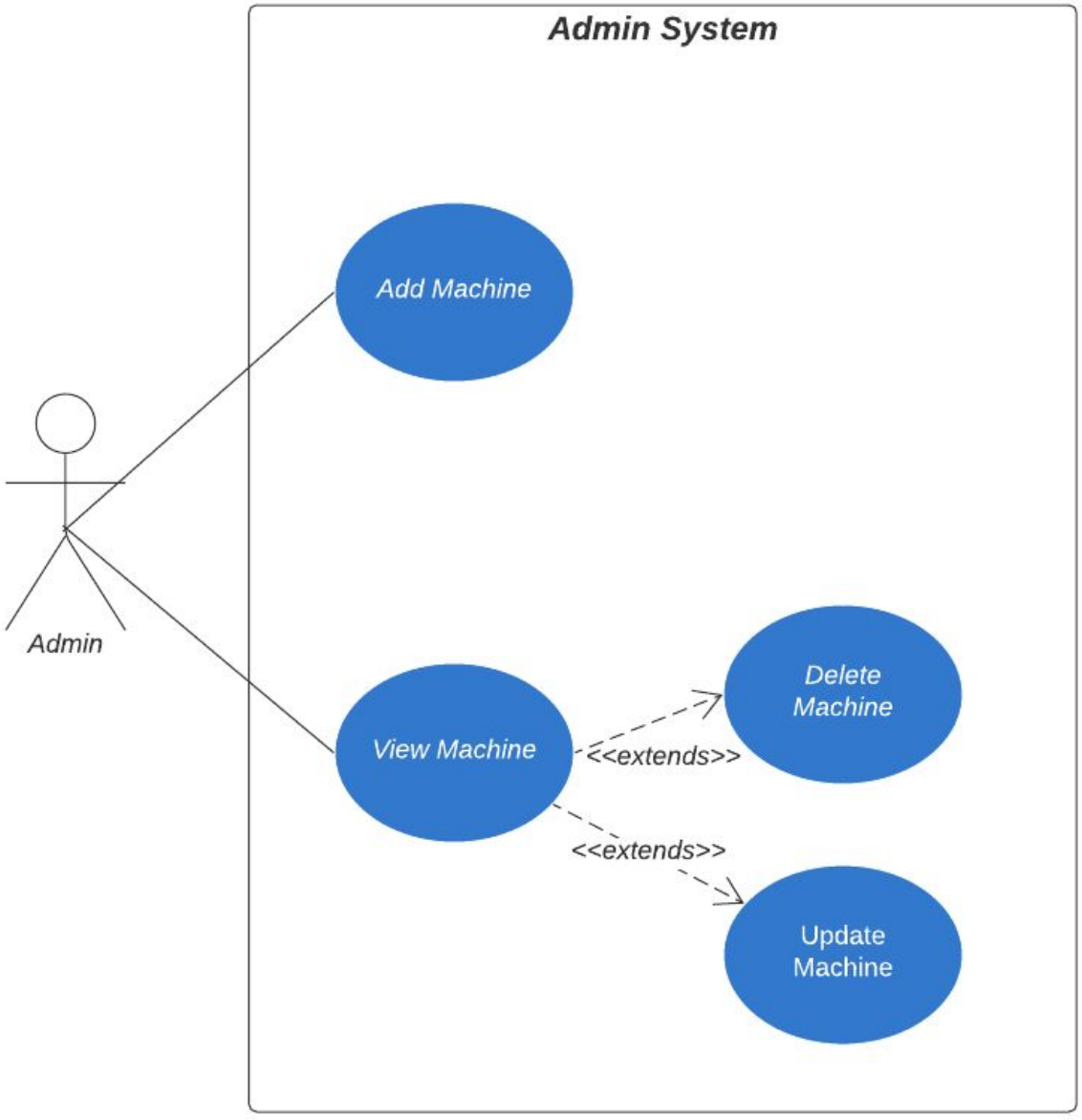
System Requirements

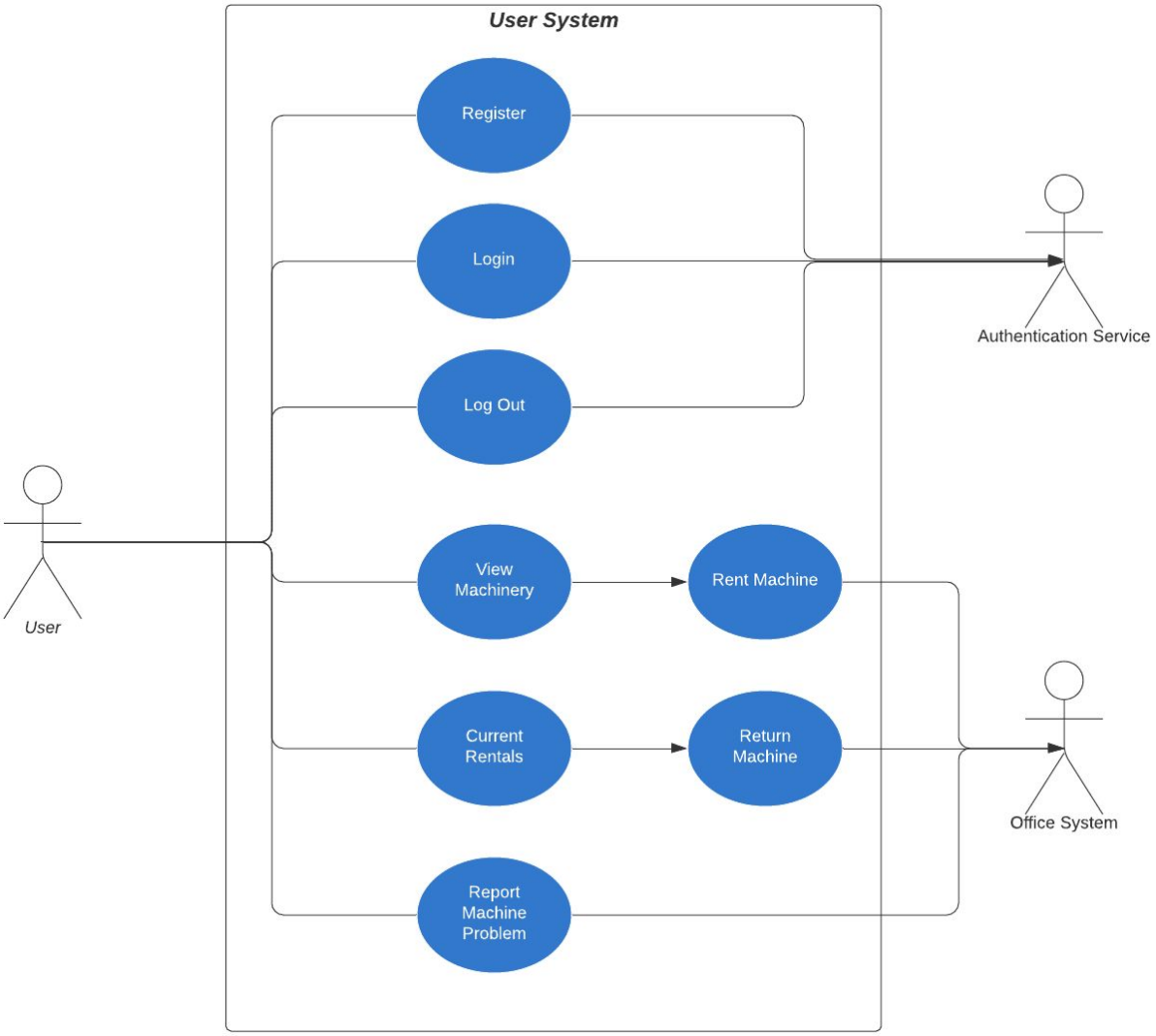
- Ensure each user who wants to rent a machine can not access the homepage of the application without having a valid user account.
- Ensure each user must be authenticated and logged in to the system before viewing the homepage.
- Provide secure login access and encrypt user credentials to help protect the user's data
- A user who has logged in can browse all the rented machinery that is available to rent in the machine rental system.
- A logged-in user can select a machine which is available to rent and fill out the required rental details needed such as length of the rental before confirming their rental and adding it to the system.
- When a user rents a machine this machine is removed from the rental browsing or its quantity available for rental is updated based on the rental.
- A logged-in user can view their current rentals in the view rental system.
- A user can be assigned administrator permissions.
- A user who is an admin can add or remove machines from the rental system.
- The rental system will provide a system for monitoring the stock for the company.
- The system must be built in line with the architectural patterns chosen and allow for an easy expansion to the code when required

Non-Functional Requirements

- The system must be easy to use and navigate for the user. The menus will be simple and clearly labelled along with an information floating button which they can click on to provide a helping note if they are unsure of what to do on a given window.
- Ensure that the system is secured and any sensitive information is encrypted
- Provide a reliable system that construction companies can depend on when they need it
- As well as have a reliable system, this system must be fast and efficient. A manager might need to book some equipment in a rush, the system must allow for, once a user has signed up, a quick, fast rental system that can be quick and easy to complete

Use Case Diagrams




User System
Register
Login
Log Out
View Machinery
Rent Machine
Current Rentals
Return Machine
Report Machine Problem
Authentication Service
Office System
User

Use Case Descriptions

Use Case 1	User Registration
Goal in Context	The user fills out the provided form with valid credentials so they can be registered with our system and login to our rental system
Preconditions	User navigates to registration
Success End Conditions	Account created and user can sign in
Failed End Conditions	Account not create and user is unable to sign in
Primary, Secondary Actors	User, application
Triggers	User clicks register button

Description	**Step**	**Action**
	1	User fills in the required form details
	2	User submits the form
	3	Form validation occurs
	4	Details are sent to the authentication service to register user
	5	User successfully registered
Extensions	**Step**	**Branching Action**
	3a	Form details are not valid and user is notified

Use Case 2	Login
Goal in Context	User fills out their login details and login to gain access to the system
Preconditions	User must have previously created an account
Success End Conditions	User logs into the system successfully and redirected to home page
Failed End Conditions	User is unable to gain access to the system
Primary, Secondary Actors	User, application
Trigger	User clicks login button

Description	Step	Action
	1	User fills in the required login details
	2	User submits the form
	3	Form validation occurs
	4	Details are sent to the authentication service see if the details are correct
	5	User successfully logged in and redirected to homepage if successful
Extensions	**Step**	**Branching Action**
	3a	Form details are not valid, the user is notified and they won't be able to log in
	4a	Authentication system has no record of this registered user, details are either wrong or they have not registered

Use Case 3	Logout
Goal in Context	User wants to log out of their account
Preconditions	User must have previously created an account and be logged in
Success End Conditions	User logs out of the system
Failed End Conditions	User is unable to log out
Primary, Secondary Actors	User, application
Trigger	User clicks logout button

Description	Step	Action
	1	User clicks the logout button
	2	User is logged out of the system and redirected to login page
	3	User successfully logged in and redirected to homepage if successful

Use Case 4	View Machinery
Goal in Context	View the available machinery that there is to rent
Preconditions	User must be logged in a click view machinery button
Success End Conditions	User is redirected to the view machinery activity
Failed End Conditions	User is unable to gain access to the activity
Primary, Secondary Actors	User, application
Trigger	User clicks view machinery button

Description	**Step**	**Action**
	1	User clicks button to view machinery
	2	User is redirected to the view machinery page

Use Case 5	Rent Machinery
Goal in Context	Go through the process to be able to rent a machine
Preconditions	User must be viewing the available machinery and click the machine they want to rent
Success End Conditions	User is redirected to the rent machinery
Failed End Conditions	User is unable to gain access to the activity
Primary, Secondary Actors	User, application
Trigger	User clicks on the available machine

Description	**Step**	**Action**
	1	User clicks on the available machine
	2	User is redirected to the rent machinery page
	3	User selects the number of weeks they want to rent the machine for
	4	User confirms the weeks and cost and is given a reference number to go to the HQ with

Use Case 6	Return Machinery
Goal in Context	User is in their current rental section and clicks on the currently rented machine to select return option
Preconditions	User must be logged in and in their current rentals section
Success End Conditions	User return is confirmed and redirected to current rentals
Failed End Conditions	User is unable to gain access to the activity
Primary, Secondary Actors	User, application
Trigger	User clicks on the machine they want to return

Description	**Step**	**Action**
	1	User clicks on the machine they want to return
	2	User must confirm their return and which site they want to return it to

Use Case 7	Add Machinery
Goal in Context	Add a new machine for rental to the system
Preconditions	Must be logged in as an Admin and have access to the admin section
Success End Conditions	New Machine is added to the system for rental
Failed End Conditions	New machine is not added
Primary, Secondary Actors	User, application
Trigger	User clicks on add machine button in admin section

Description	Step	Action
	1	User clicks button to add machine
	2	User enters the form details required to add a new machine
	3	Machine details are validated
	4	Machine is added to the database and available for rental

Use Case 8	Remove a Machine
Goal in Context	Removes a machine from the database and rental
Preconditions	Must be logged in as an Admin and have access to the admin section
Success End Conditions	Machine is removed from the rental system
Failed End Conditions	Machine is not able to be removed
Primary, Secondary Actors	User, application
Trigger	User clicks the remove icon on the machinery inventory list

Description	Step	Action
	1	User clicks button to remove a machine
	2	User must confirm their selection
	3	Machine is removed from the database and rental system

Quality Attributes & Tactics

We decided to use Java for the project as it was firstly a language the majority of us were familiar with. It will allow for developing intuitive UIs and setting up storage of users and machines with relative ease. In addition Java supports portability as Java byte code is able to run on any hardware using it's Java Virtual Machine (JVM).

JavaFX to create the UIs will allow for easy creation of UIs that are both intuitive and easily implemented into the application.

Security

To ensure security will be encrypted with admin accounts having additional levels of encryption so despite the application running locally the end user doesn't see the plain text form of any passwords. Defensive programming should also be used whenever possible.

Extensibility

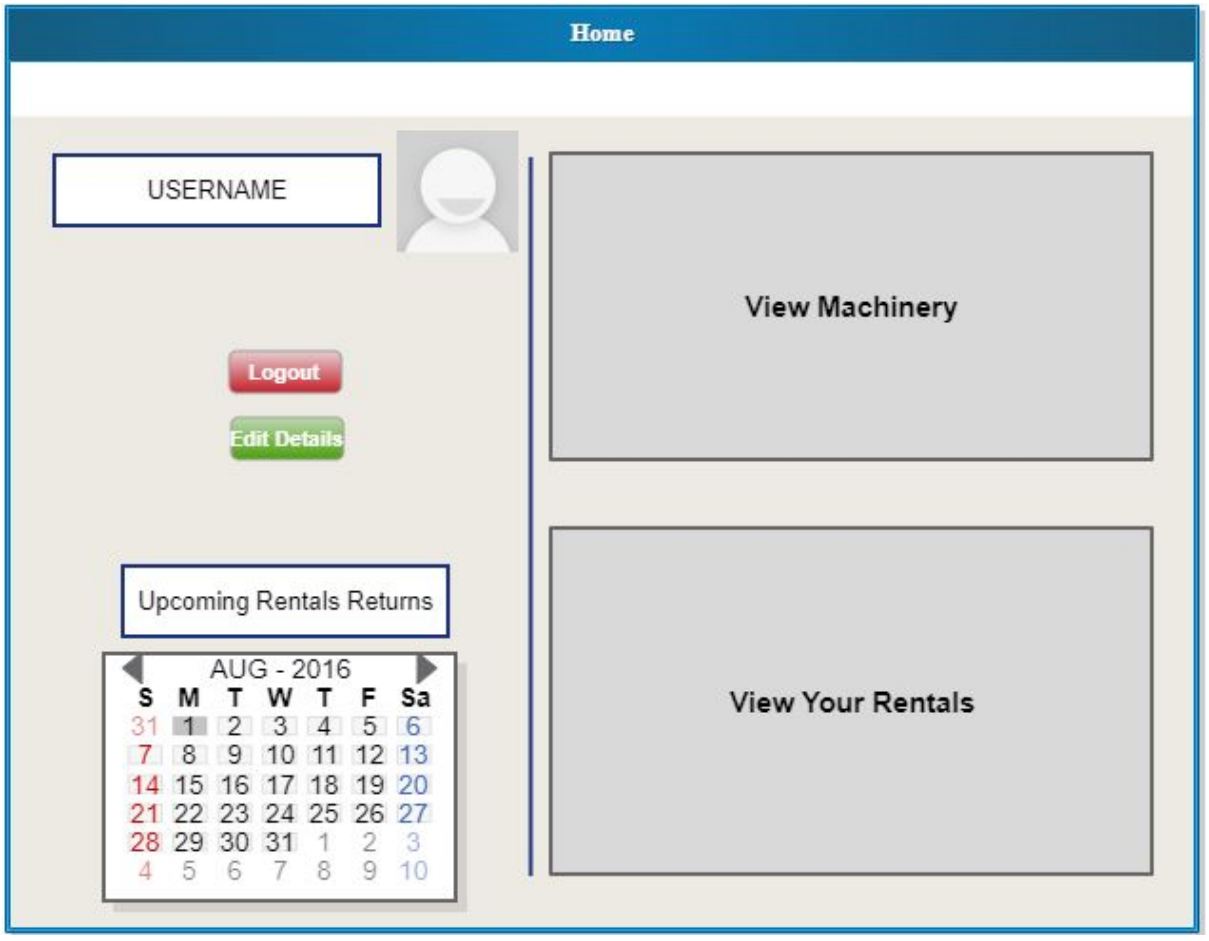
This application will consist of a java application which runs locally (no server). This software will be designed with the goal of being able to add functionality in the future in the future, for example: The "AlertBox" class when called requires a title and message. This can fit almost any alert required in the future and will not affect other parts of the program in a negative way.

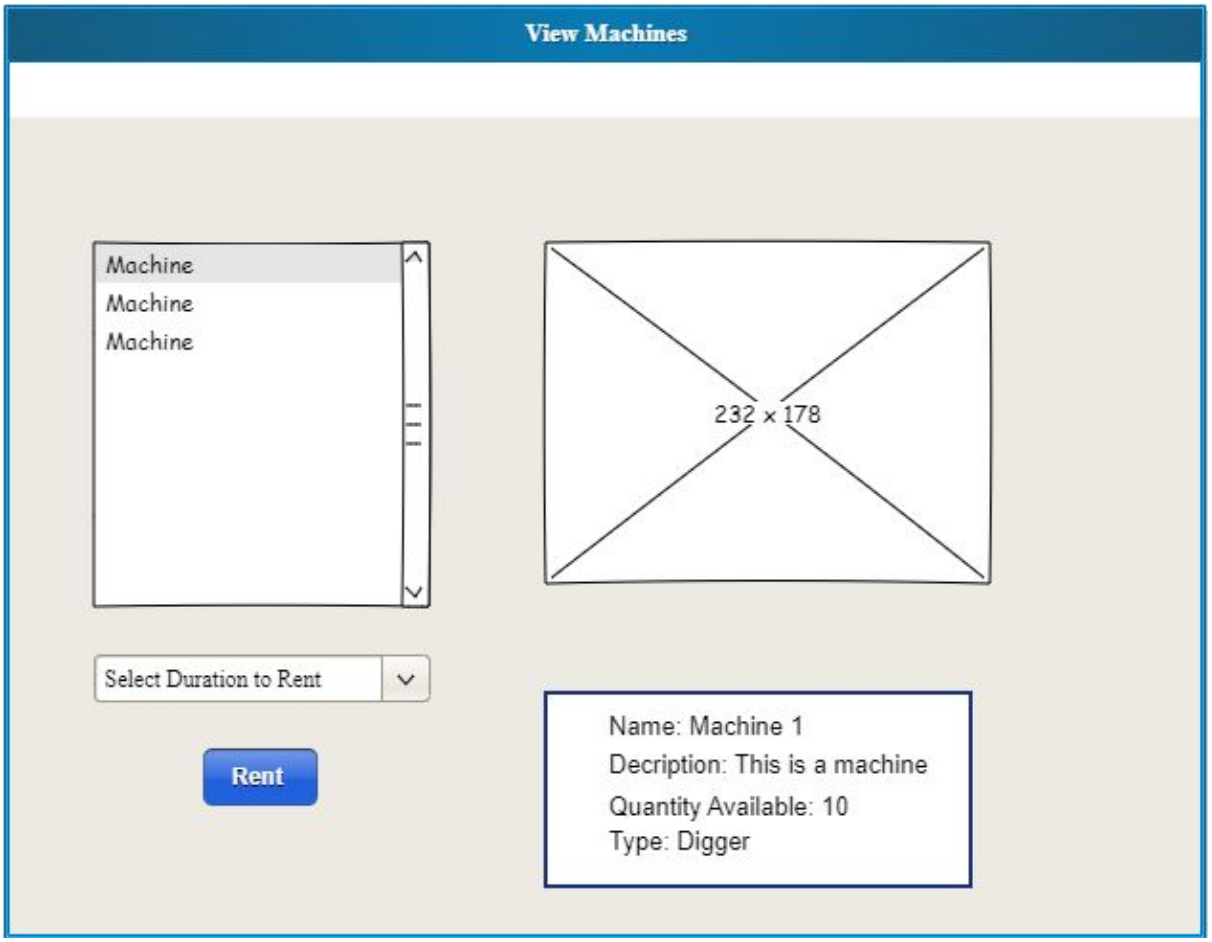
Maintainability

Keeping the software organized and accessible is important to correct bugs and makes it easier for new people working on the project to understand the code. For our project we will break it into 3 main packages, Authentication, Home and Runner.

GUI Prototypes

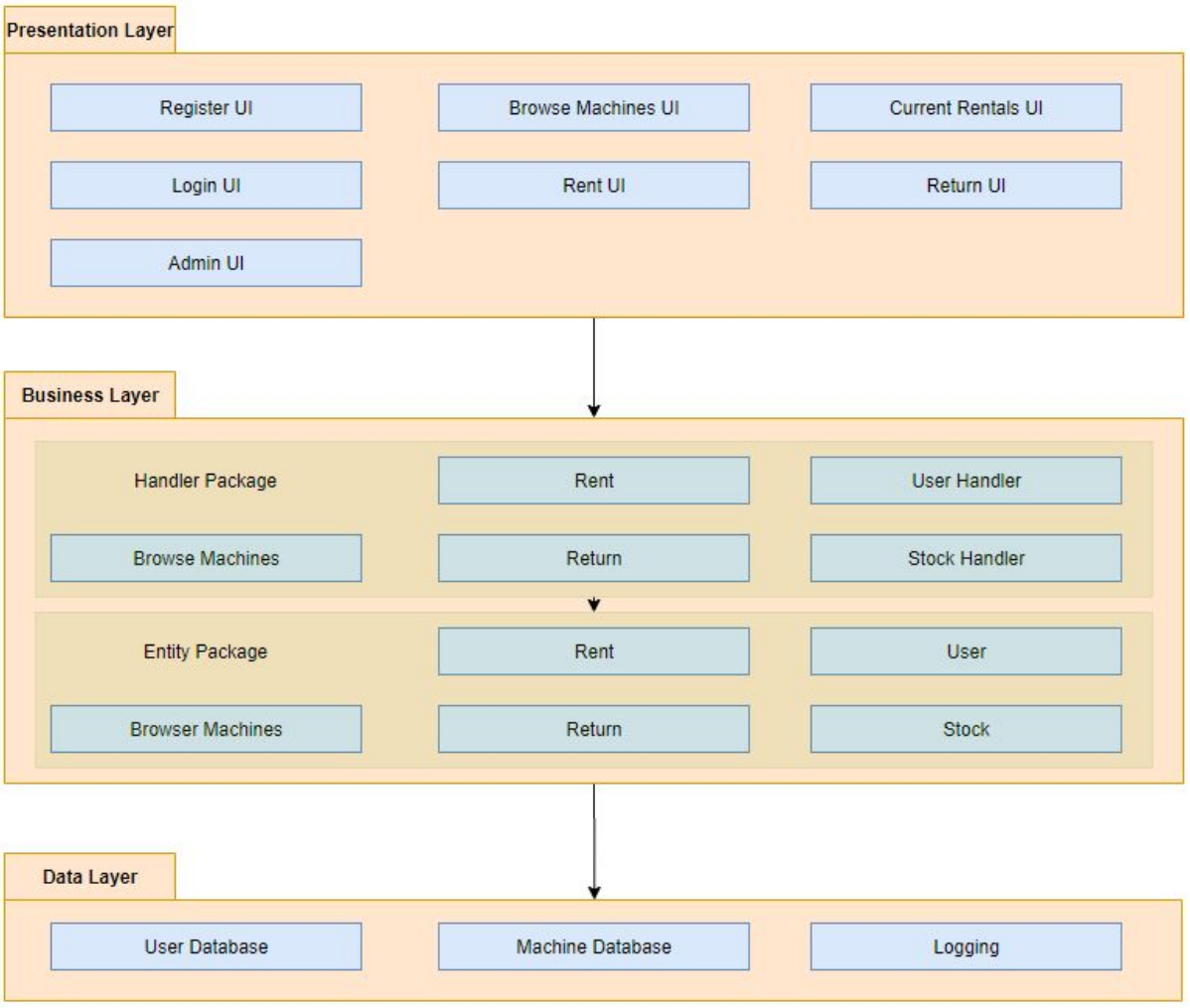
Below we have a view of what our GUIs might represent in our finished product. We used a GUI builder framework called Pencil to create these prototypes. The first prototype GUI is of the main home page for a user. From this you can get an idea of the style and layout of the home page. The second GUI is of the rental page where a user can flick through a list of the available machines and see the details available for each one before renting for a required amount of time.




View Machines
Machine
Machine
Machine
232 × 178
Select Duration to Rent
Rent
Name: Machine 1
Decription: This is a machine
Quantity Available: 10
Type: Digger

System Architecture

Package Diagram



Discussion:
As you can see above our Architecture has been split into 3 layers for this project. The Presentation layer (UI/View), the business layer (Handles logic and controllers) and the Data layer (Database System). This architecture follows the design pattern known as a model-view-controller or MVC for short. This design allows for easy changing of system components if there is a change in requirements or additions requirements are added.

Analysis Sketches

Candidate Objects (Data-Driven Design)

This section identifies a list of candidate objects derived using Data Driven Design (DDD).

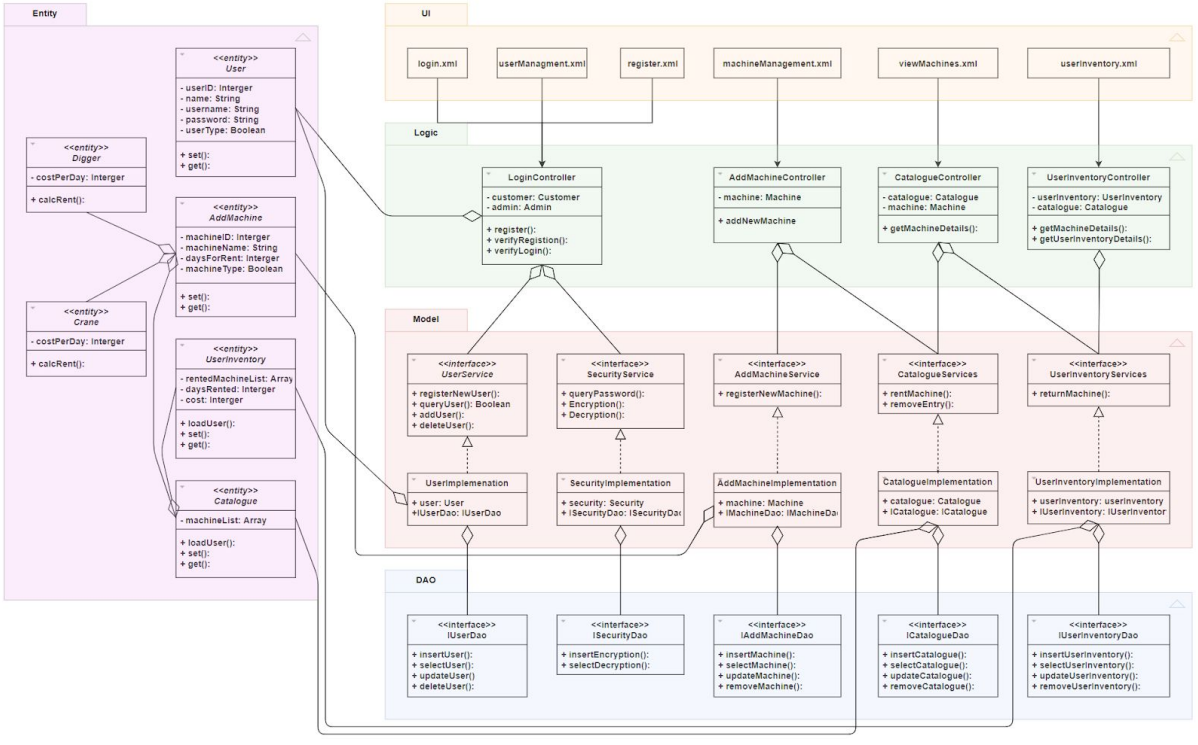
User	UserType	Rent	Machine	Node
Admin	Login	PickupDate	MachineDetails	Software
Customer	Logout	Username	Cost	System
Register	AccountLogin	Password	Location	Management

Legend:

	Class that will be implemented
	Class functionality covered by implemented class
	Part of meta language
	Ambiguous class

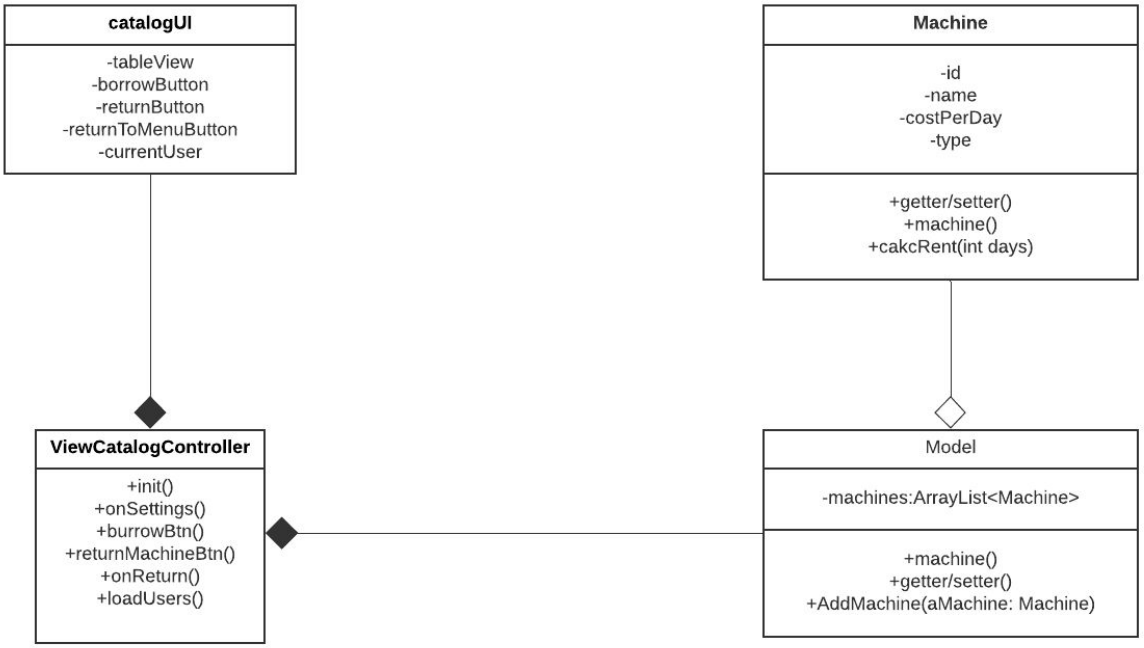
Analysis Class Diagram

The purpose of this diagram is to provide analysis and design of the static view of an application, to describe the responsibilities of a system, and to use as a base for forward and reverse engineering. The main associations are aggregation along with inheritance. This diagram builds upon the use cases, and the requirements.

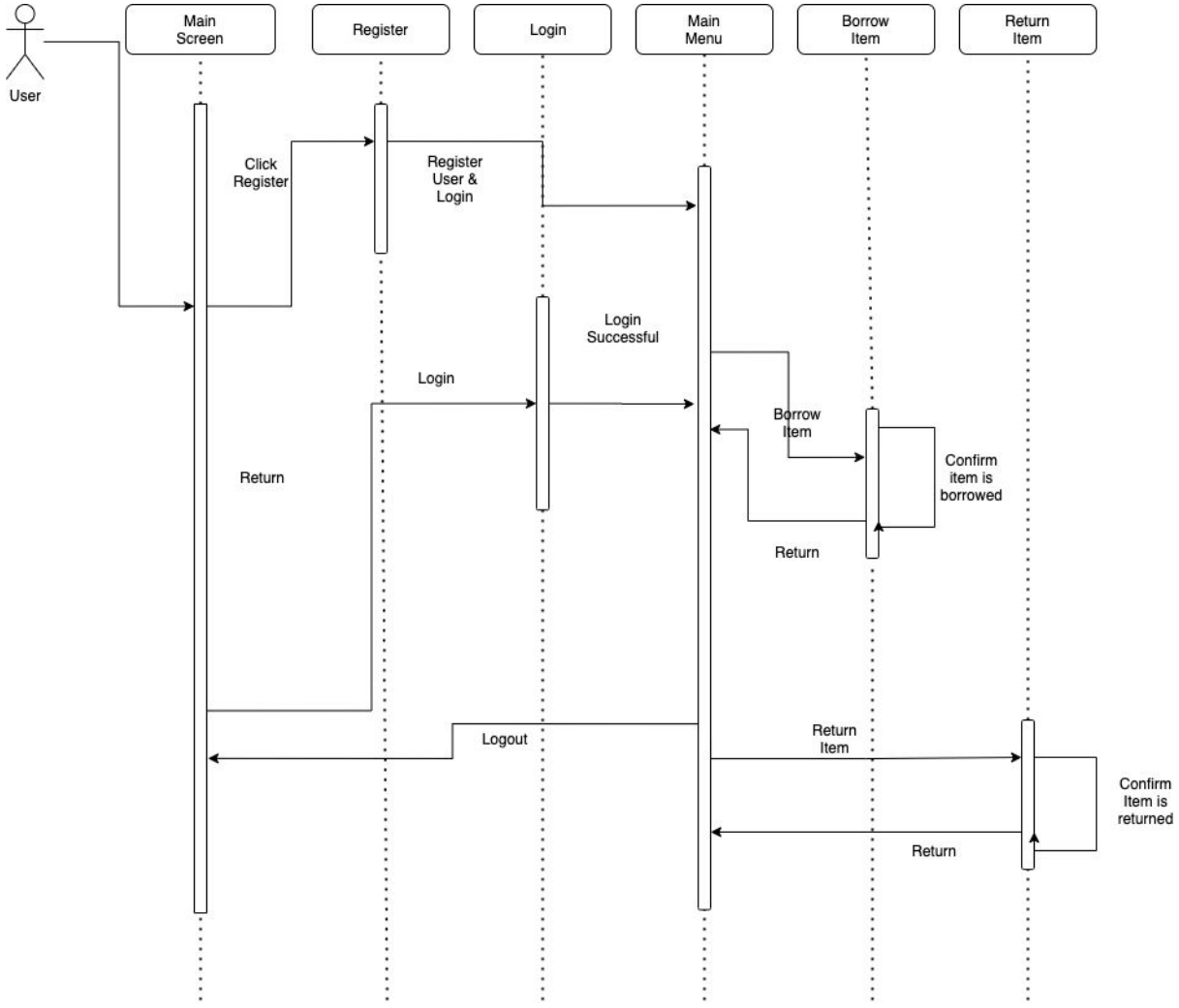


MVC Class Diagram

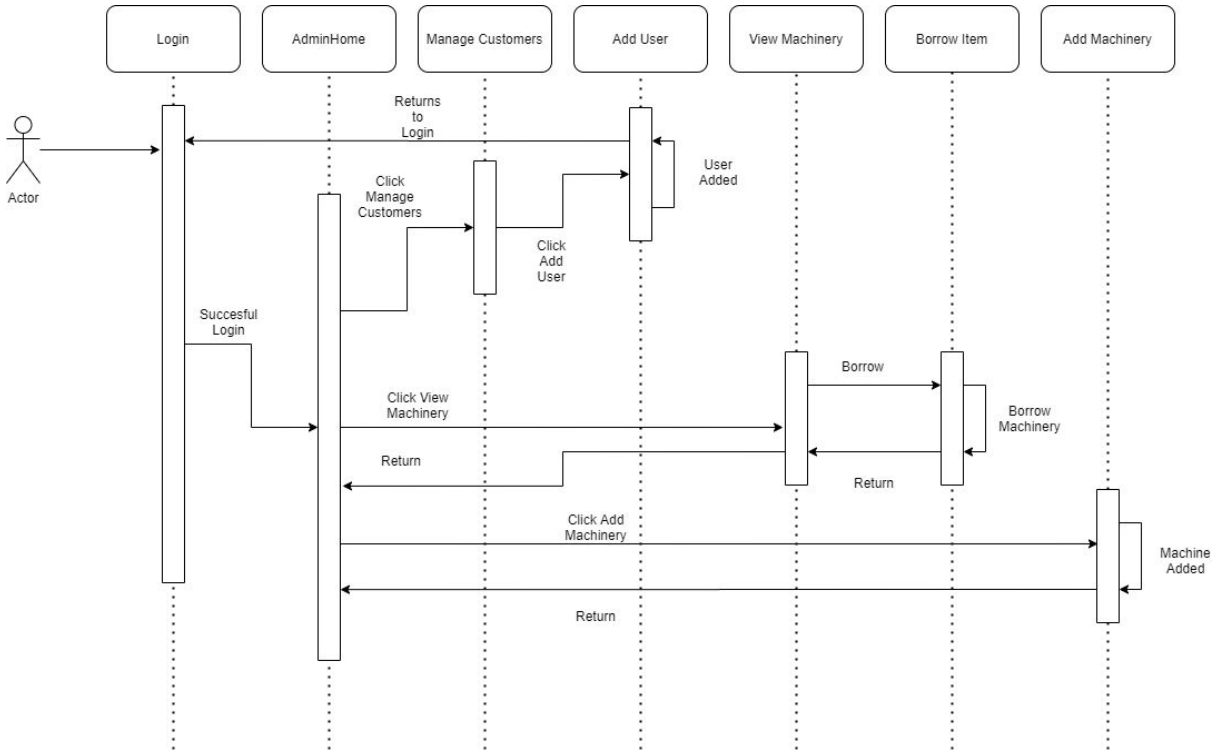
This is the MVC class diagram for viewing which machines are available for rental. The user can scroll through the list of machines and if the machine they want is available they can click the catalogUI borrowButton and continue to the rental confirmation.



Sequence Diagrams

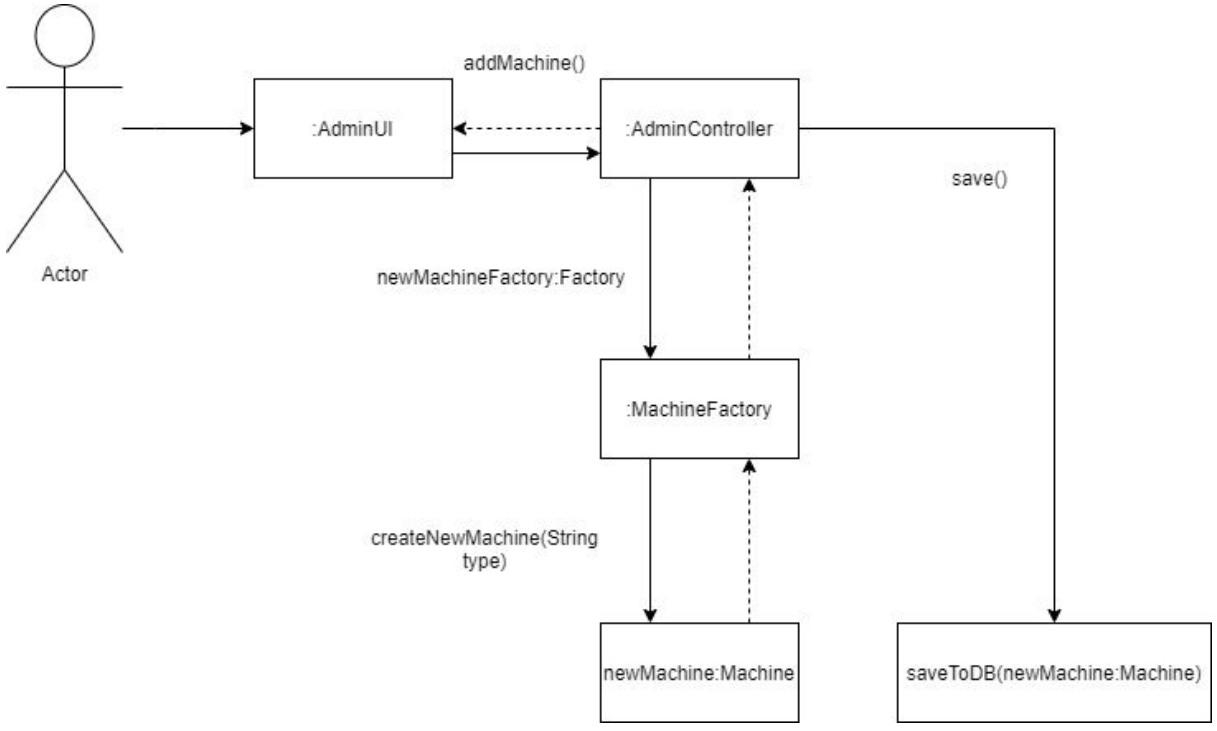


Above sequence diagram is for a standard user account.



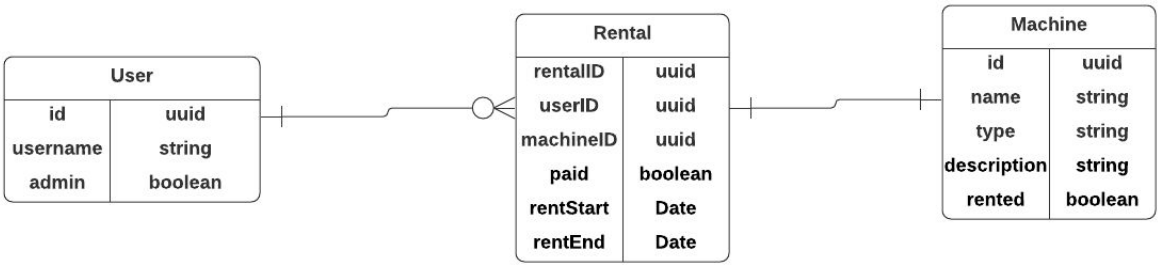
Above sequence diagram is for an admin account type.

Communication Diagram



Communication diagram for admin creating new machine with factory method also

Entity-Relationship Diagram



Code Breakdown

Number of Classes: 34 classes
Number of Packages: 8 packages

Tabular Class Listing

Table breakdown of code, LOC is an approximation in some cases.

Package	**Class/File Name**	**LOC**	**Author**
sample	Main	81	Group
sample	Navigation	29	V.K, M.M
sample	Statics	12	V.K, C.C
sample	AlertBox	34	P.B
Auth>Logic			
sample.Authentication.Logic	FileManager	83	V.K, C.C, E.F
sample.Authentication.Logic	IValidator	6	V.K
sample.Authentication.Logic	LoginController	148	Group
sample.Authentication.Logic	PasswordValidator	18	E.F, V.K
sample.Authentication.Logic	RegisterController	203	P.B, V.K, C.C, M.M
sample.Authentication.Logic	Validator	7	E.F, V.K
Auth>Model			
sample.Authentication.Model	AccountType	3	V.K
sample.Authentication.Model	Admin	47	V.K, E.F, M.M, C.C
sample.Authentication.Model	Customer	36	V.K, E.F, C.C, M.M
sample.Authentication.Model	Decryption	44	E.F, V.K, M.M
sample.Authentication.Model	Encryption	45	V.K, E.F, M.M
	ISecurity	6	V.K
sample.Authentication.Model	Security	13	V.K
sample.Authentication.Model	User	109	V.K, C.C, M.M

sample.Authentication.Model	UserAdapter	77	V.K, C.C
Auth>UI			
	login.fxml	29	V.K.
	register.fxml	48	V.K, M.M
Command			
sample.Command	ICommand	5	V.K
sample.Command	NavigationInvoker	8	V.K
sample.Command	Navigator	5	V.K
sample.Command	Previous	9	V.K
Home>Logic			
sample.Home.Logic	AddMachineController	101	M.M, E.F, V.K, C.C
sample.Home.Logic	AddUserController	85	P.B, M.M
sample.Home.Logic	AdminHomeController	94	M.M, V.K, E.F, C.C
sample.Home.Logic	BorrowedItemsController	144	C.C, M.M, E.F
sample.Home.Logic	ManageCustomers	68	P.B, V.K, E.F, M.M
sample.Home.Logic	UserHomeController	89	C.C, V.K, E.F, M.M
sample.Home.Logic	ViewCatalogController	144	C.C, P.B, E.F, M.M
Home>Model			
sample.Home.Model	Crane	32	C.C, M.M
sample.Home.Model	Digger	35	C.C, V.K, M.M
sample.Home.Model	Machine	52	C.C, V.K, E.F, M.M
sample.Home.Model	MachineAdapter	49	C.C, V.K, M.M
sample.Home.Model	MachineFactory	21	C.C, M.M, V.K
Home>UI			
	addMachine.fxml	42	E.F, V.K

	addUser.fxml	44	P.B
	adminHome.fxml	89	V.K, C.C
	borrowedItems.fxml	71	V.K, C.C
	catalog.fxml	67	C.C, V.K, E.F
	manage_customers.fxml	47	V.K, P.B
	userHome.fxml	83	V.K, C.C
Runner>Logic			
sample.Runner.logic	LenderController	91	V.K, E.F
Runner>UI			
	lender.fxml	28	V.K
Runner			
sample.Runner	IAdapter	7	V.K
Test			
sample.Test	CraneRentTest	33	M.M
sample.Test	EncryptionTest	36	M.M

Total Code Developed

A total of approximately 2,650 lines of code was written in this project

Team Member Contribution

Note: Lines added also include lines created using the SceneBuilder of JavaFX and "out" folder.

Team Member	**Lines Added (Approximation)**
Conor Canton	4700
Vainqueur Kayombo	4700
Panos Brennan Andreou	880
Eoin Flynn	320
Murdo Mackenzie	360

Code

Coding Fragments

RegisterController.java

```
public void manualAdmin(String Name, String Username, String Password) {
    String time = String.valueOf(System.currentTimeMillis());

    User origin = new Admin(time, Name, Username, Password, AccountType.ADMIN, emptyMac);
    origin.encryptPassword();
    users.add(origin);
    io.serializeToFile( path: "AdminDB.ser", users);
}
```

```
public void manualUser(String Name, String Username, String Password) {
    String time = String.valueOf(System.currentTimeMillis());
    regUser = new Customer(time, Name, Username, Password, AccountType.CUSTOMER, emptyMac);

    regUser.encryptPassword();

    users.add(regUser);
    Statics.CurrentUser = regUser;

    io.serializeToFile( path: "CustomerDB.ser", users);
}
```

AlertBox.java

```java
public class AlertBox {

    public static void display(String title, String message){
        Stage window = new Stage();

        window.initModality(Modality.APPLICATION_MODAL);// Stops input into other windows.
        window.setTitle(title);
        window.setMinWidth(250);
        window.setMinHeight(125);

        Label label = new Label();
        label .setText(message);

        Button closeButton = new Button( s: "Close the window");
        closeButton.setOnAction(e -> window.close());

        VBox layout = new VBox( v: 10);
        layout.getChildren().addAll(label, closeButton);
        layout.setAlignment(Pos.CENTER);

        Scene scene = new Scene(layout);
        window.setScene(scene);
        window.showAndWait();// Waits for window to be closed then returns to previous window.


    }
}
```

AddUserController.java

```java
public void onAdd(ActionEvent actionEvent) {
    try {
        String username = userName.getText();
        String name = nameText.getText();
        String pw = password.getText();
        String type = accountType.getValue().toString();
        RegisterController rc = new RegisterController();
        //if else statements to check if account type is valid

            if(type.equals("Admin")) {
                rc.manualAdmin(name, username, pw);
            } else if (type.equals("User")) {
                rc.manualUser(name, username, pw);
            }
            Main.currentStage.setFXMLScene( sceneID: "Authentication/UI/login.fxml", new LoginController());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

LoginController.java

```
public void loginButtonOnAction(ActionEvent event) throws IOException {
    if (!usernameField.getText().isBlank() && !passwordField.getText().isBlank()) {
        if (validateLogin(usernameField.getText(), passwordField.getText())) {
            messager.setText("Logged in as: "+Statics.CurrentUser);
            ArrayList<User> users= new ArrayList<>();
            users.add(Statics.CurrentUser);
            messager.setStyle("-fx-text-fill: green;");
            io.seria   sample.Authentication.Logic.LoginController
                       private Label messager

            if (Statics.CurrentUser.getType() == AccountType.CUSTOMER) {
                Main.currentStage.setFXMLScene( sceneID: "Home/UI/userHome.fxml", new UserHomeController()); //test 2 Home/UI/adminHome.fxml
            } else {
                Main.currentStage.setFXMLScene( sceneID: "Home/UI/adminHome.fxml", new AdminHomeController()); //test 2
            }
        }
    } else {
        messager.setText("Please enter a Username AND Password");
        messager.setStyle("-fx-text-fill: red;");

    }
}
```

PasswordValidator.java

```
public PasswordValidator(int version) {
    switch (version) {
        case 0: this.passPattern = "(?=.*[a-z])(?=.*[A-Z])(?=\\S+$).{6,}"; break;
        default: this.passPattern = "(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=\\S+$).{6,}"; break;
    }
}
```

Design Patterns

Factory Pattern

```
public class MachineFactory {
    public Machine createNewMachine(String type) {
        if (type == null || type.isEmpty())
            return null;

        if ("Digger".equals(type)) {
            System.out.println("CREATING DIGGER TYPE");
            return new Digger();
        } else if ("Crane".equals(type)) {
            System.out.println("CREATING CRANE TYPE");
            return new Crane();
        }

        return null;
    }
}
```

```
public void onAdd(ActionEvent actionEvent) {
    try {
        String type = machineType.getValue().toString();
        Machine newMachine = factory.createNewMachine(type);
```

In our implementation we made use of the design pattern known as the factory method. As we want to create different types of machines and in the future we may want to add extra support for new machine types. This can be easily done thanks to the Factory Method. We pass in via a drop down box what type of machine we are creating for the database. This machine type is then passed into the Machine Factory and returns a machine of the type we wanted. If we add new machine types to the models we can just continue the else statement to allow for the creation of this new machine object.

Observer Pattern

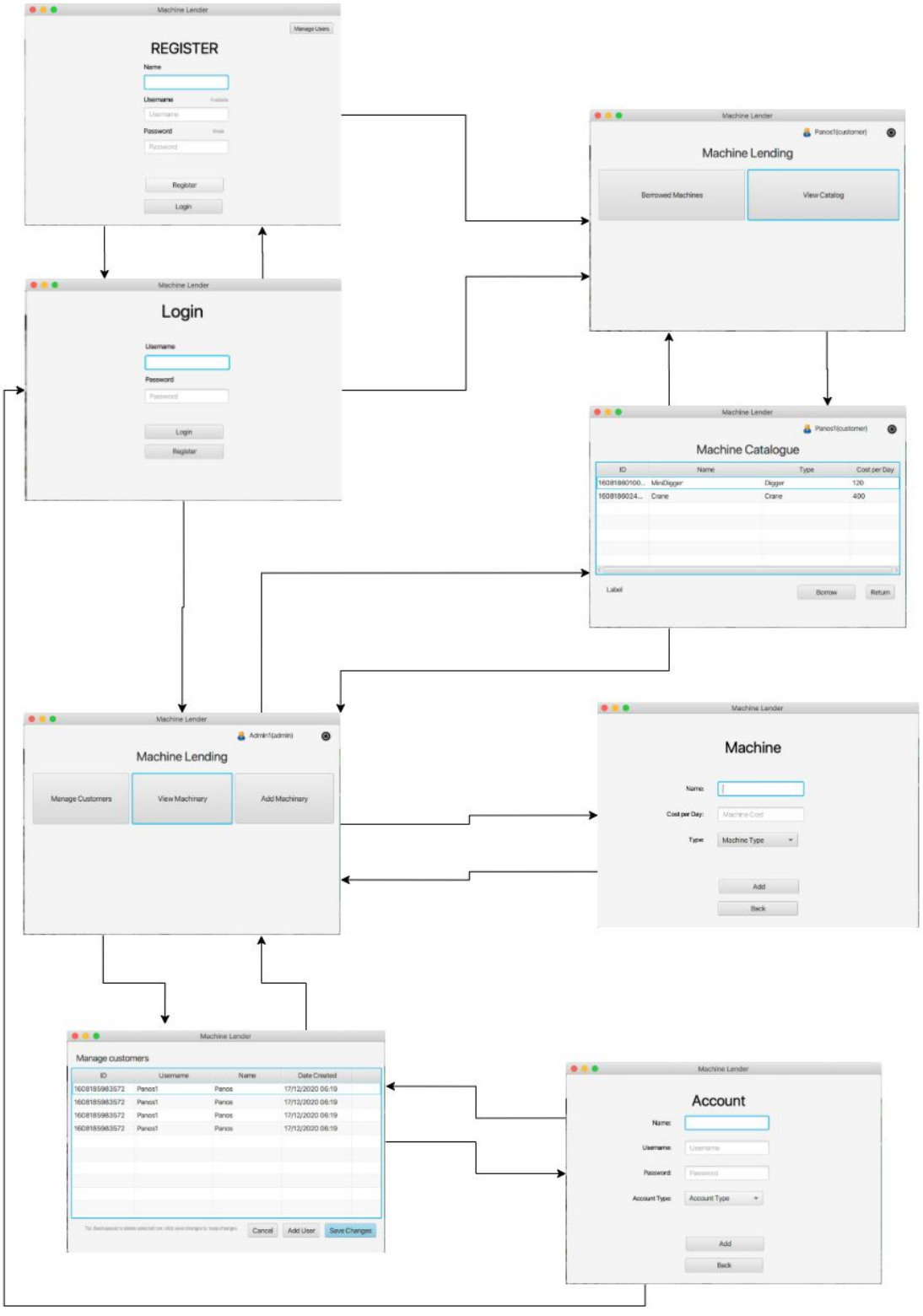
```
public class InventoryManager implements Subject{
    //This class will manage the behaviors of all the observes, the observers
     ArrayList<Observer> observers;
    int inventory=0;
   public  InventoryManager() { observers=new ArrayList<>(); }
    @Override
    public void register(Observer newObserver) { observers.add(newObserver);

    @Override
    public void unregister(Observer oldObserver) {
       int observerIndex= observers.indexOf(oldObserver);
       if(observerIndex>-1)
           observers.remove(observerIndex);
        System.out.println("Observer deleted: ");
    }

    //notifies all servers of any changes within the inventory
    //changing their values every where;
    @Override
    public void notifyObserver() {
       for(Observer o : observers)
           o.update(inventory);
    }

    public int getInventory() { return inventory; }

    public void setInventory(int inventory) {
        this.inventory = inventory;
        //when the inventory changes all obsers are notified
        notifyObserver();
    }
}
```

We used the observer pattern to listen out for any changes in the user's inventory and when changes occur then we can notify the observer. So we can update the inventory.

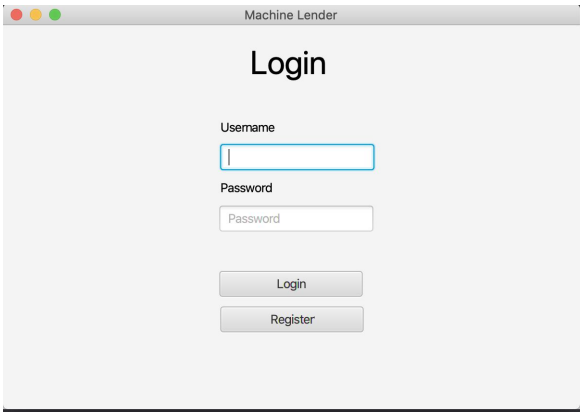
Abstract Machine Class

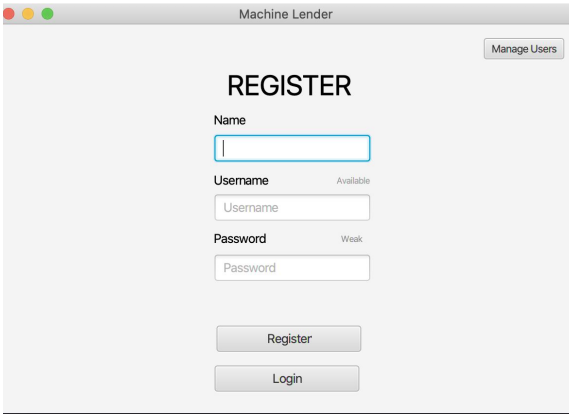
GUI Screenshots

OverView/Navigation



Login/Register



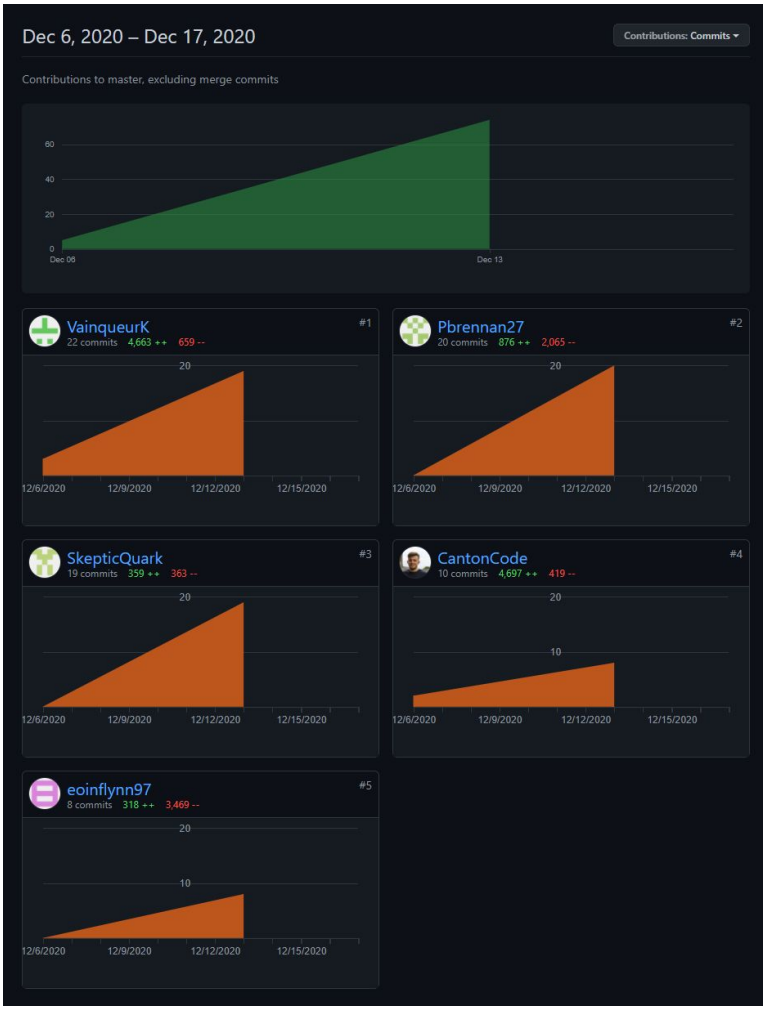


Added Value

Version Control - Github

https://github.com/VainqueurK/lender
We used Github for version control in this project. Github is a web-based hosting service for version control using Git. It allowed us to track and compare changes as well as recover states in the event bad code was pushed or things were changed accidentally. All group members attempted to push regularly to reduce integration problems. Before pushing code it was decided to pull and merge code locally. This allowed us to check for errors before pushing the code.

Below is a screenshot of the commits and line additions and deletions in the repository:



JUnit Tests

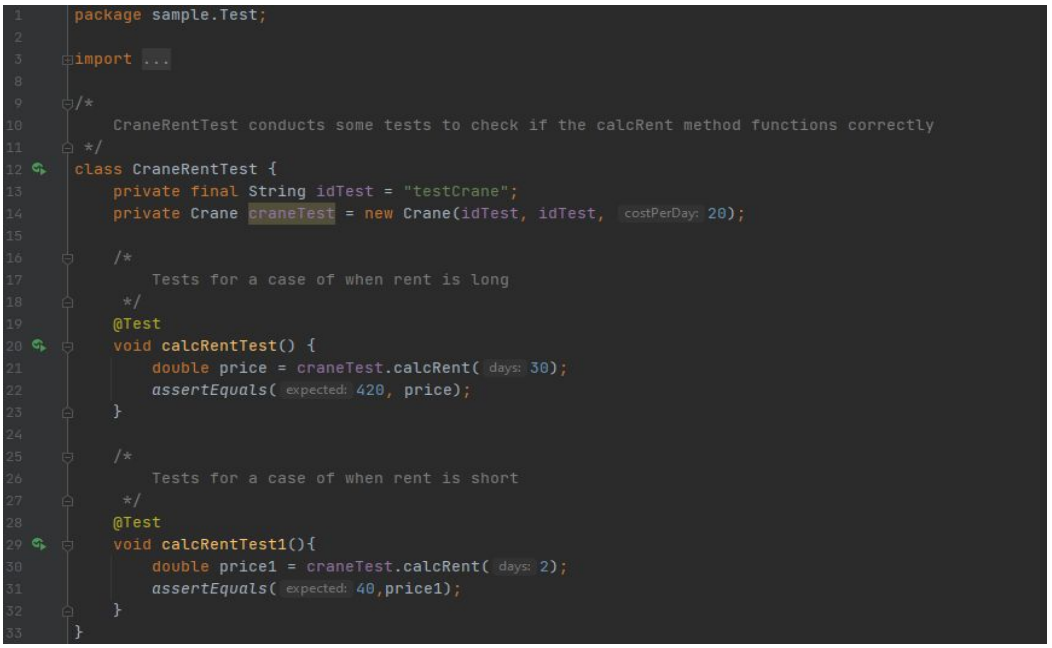
JUnit 5 was used to test code during development. Using unit tests allowed us to test the functionality of our code to test different situations and make the development process easier.

EncryptionTest

```java
package sample.Test;

import org.junit.jupiter.api.Test;
import sample.Authentication.Model.Encryption;
import sample.Authentication.Model.Decryption;
import static org.junit.jupiter.api.Assertions.*;

/*
    EncryptionTest tests the functionality of encrypting and decrypting user and admin passwords for added security
 */
class EncryptionTest {
    private Encryption encryptionTest = new Encryption();
    private Decryption decryptionTest = new Decryption();
    private final String passwordTest = "Test123";
    private final String expected = "Test123";

    /*
        Tests if layer one encryption works as expected by encrypting and decrypting a test password
     */
    @Test
    public void passwordSecurityTest(){
        String encrypted = encryptionTest.layerOne(passwordTest);
        String actual = decryptionTest.layerOne(encrypted);
        assertEquals(expected, actual);
    }

    /*
        Tests if layer two encryption works as expected by encrypting and encrypting a test password
     */
    @Test
    public void passwordSecurityTest1(){
        String encrypted1 = encryptionTest.layerTwo(passwordTest);
        String actual1 = decryptionTest.layerTwo(encrypted1);
        assertEquals(expected, actual1);
    }
}
```

Above is the EncryptionTest class which tests the functionality of encrypting and decrypting passwords. Making sure this code functioned correctly is important for user and admin security. Test cases were given a password and told to encrypt and decrypt it, in one case using the layerOne encryption and in another using layerTwo.

CraneRentTest



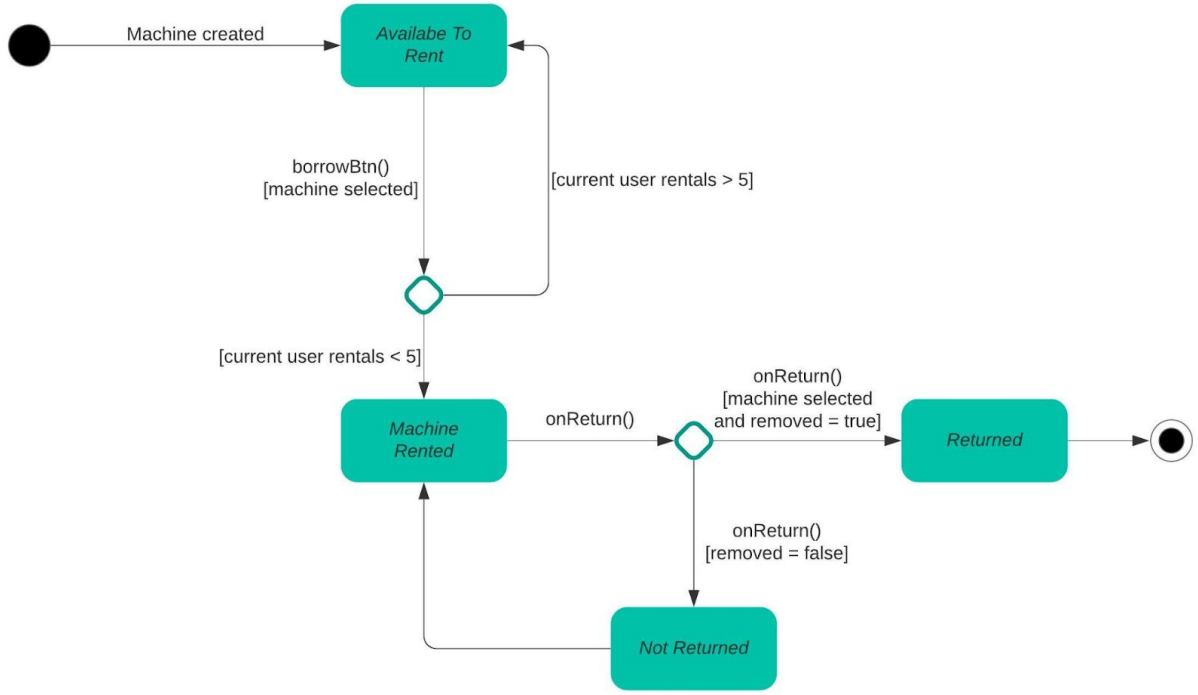
```
package sample.Test;

import ...

/*
    CraneRentTest conducts some tests to check if the calcRent method functions correctly
 */
class CraneRentTest {
    private final String idTest = "testCrane";
    private Crane craneTest = new Crane(idTest, idTest,  costPerDay: 20);

    /*
        Tests for a case of when rent is long
     */
    @Test
    void calcRentTest() {
        double price = craneTest.calcRent( days: 30);
        assertEquals( expected: 420, price);
    }

    /*
        Tests for a case of when rent is short
     */
    @Test
    void calcRentTest1(){
        double price1 = craneTest.calcRent( days: 2);
        assertEquals( expected: 40,price1);
    }
}
```

Above is the CraneRentTest which created a test crane and then checked if the rent calculations were correct for different cases, in one case for a short term rent and in the other for a longer term rent.

Recovered Architecture and Design Blueprints

Architectural Diagram

State Chart (Rental Model)



Critique and Analysis of Design Artifacts

From a general perspective we are happy with how we have implemented our project. As our group started later than others working on a tighter time schedule made both completing diagrams and completing the implementation much more challenging. While we didn't get to finish the project as much as we would have liked, we were able to implement most of the use cases required and we were also able to design our code so that adding additional functionality in the future is not difficult.

Our final implementation differed slightly from our original diagrams that we developed during the Analysis phase. These diagrams did, however, help us find the structure we went with for our final design and greatly influenced our thinking during the implementation.

Had we the opportunity to go and do this project again on an equal timeline to other groups we could have tried to implement other features such as a server side aspect using a Firebase database for example to store user and machine data. Beginning to code sooner on in the timeline also would have also helped implementation and given more time to do recovered architecture and design blueprints.

References

How to add JavaFX to IntelliJ: https://openjfx.io/openjfx-docs/#IDE-Intellij

Java JDK: https://www.oracle.com/java/technologies/javase-jdk11-downloads.html

IntelliJ Download: https://www.jetbrains.com/idea/download/#section=windows

StackOverflow - How to install JavaFX in IntelliJ:
https://stackoverflow.com/questions/53668630/how-to-run-javafx-applications-in-intellij-idea-ide

How to create JUnit tests in IntelliJ:
https://www.jetbrains.com/help/idea/testing.html#add-testing-libraries