Software Requirements Specification (SRS)
Project 1

Authors: Nathaniel Finley, Alex Lepird, Jack Mansueti, Caleb Sherman, Hansheng Zhao

Customer: General Motors

Instructor: James Daly

1 Introduction

The Software Requirements Specification (SRS) document is a detailed description of the requirements for General Motors Automated Report Generation system. The system automatically generates reports; highlighting the paint defects on cars that come from the assembly line. The SRS contains these sections:

The first section gives an introduction and identifies the purpose and scope of the software system as well as a list of terms with their respective definitions. The second section provides an overall description of the software system and examines the product perspective, product functions, and user characteristics. This subsection also covers the constraints, assumptions, dependencies, and apportioning of the requirements for the system. The third section outlines the specific requirements for the software system in a hierarchical fashion. The fourth section identifies and demonstrates each use case for the system and visually represents this in a class diagram along with sequence, state, and use case diagrams. The fifth section gives an overview of the prototype and describes what the prototype will show of system functionality as well as how to run the prototype. This section will also provide the URL for the prototype and sample scenarios for system functionality. The sixth section will provide a list of documents referenced within the SRS.

1.1 Purpose

The purpose of the SRS document is to provide a detailed description of a software system being created for General Motors. The system will help automate General Motors process of documenting paint defects in cars coming off the assembly line. The intended audience of the SRS are the developers of the system. In addition to the developers, the customer and clients can provide minor input via the SRS document. The SRS document provides context for the system as well as the architecture of the actual system itself.

1.2 Scope

The software system being developed is an automated paint defect report generator. The application will adopt the following workflow: The GM client analyst enters paint defect data (see section 4 for detailed defect data) into the software system. The defect data is placed into a database alongside other recorded defects. GM client analyst attempts to retrieve a defect data report based on a specific time window. The system creates a report based on the requested criteria and returns it to the GM client analyst. The new system implements a method that replaces the previous method which required the analyst to manually document the location, type, and severity of each defect, compiles the defects following a certain time window, and generates paper reports.

The system will be a web-based application; this allows for access anytime and anywhere. The GM client analyst will be required to provide their GM employee credentials before using any feature on the system.

Note that the system will not automate the information gathering process. The original project specification (9) from the customer states “The client would like to automate the system to eliminate the paper diagrams and automatically generate the desired reports from the entered data.” The system will do this, but will not contain additional features.

1.3 Definitions, acronyms, and abbreviations

Term	Definition
Client Analyst	Employee at GMC that examines vehicles for flaws in the paint and finish and creates paper diagrams.
DB	Database

DPU	Defects per unit
GM	General Motors
GMC	General Motors Company
Sev	Severity of defect
Point Defect	Point defect is a paint defect on a specific location of the car (noted by a point on the car diagram)
Software Requirements Specification (SRS)	A document that completely describes all of the functions of a proposed system and the constraints under which it must operate.
SQL	Structured Query Language Used for database management.
Frontend	A web application’s web page that the user can interact with.
Backend	The server process that handles user requests and generate response accordingly.
PHP	PHP: Hypertext Preprocessor, a popular language for building web applications.
SQLite	An embedded database engine that provides SQL language interface and database functionalities.
GDBM	GNU DBM implementation, a database engine for efficiently storing arbitrary key value pair.
JavaScript	A programming language that can run in the browser or on the server.
Python	A dynamic language that is suitable for building prototypes and production software.
SSL	Secure Socket Layer, a protocol for secure and encrypted browsing.
HTTPS	HTTP protocol that utilize SSL for security.
FreeBSD	A UNIX-like system that is both stable and secure.

REST	Representational state transfer, a method for communications between computer systems.
NodeJS	An efficient JavaScript interpreter for running JavaScript on servers.
API	Application Programming Interface, the interface that applications provide for developers to use to build other applications.
URL	Universal Resource Locator, a link that uniquely identifies a resource on the internet.

1.4 Organization

The remaining sections of the SRS contain a detailed description of the system, specific and modeling requirements, and class, state, data dictionary, and use case diagrams.
The remaining organizational structure of the SRS:

1. Overall Description
2. Specific Requirements
3. Modeling Requirements
4. Prototype Overview
5. References
6. Point of Contact

2 Overall Description

The following sections will cover the internal and external factors that play an important role in the operation of the product. The sections will also give a better understanding of product functionality and the constraints of the software system.

2.1 Product Perspective

Currently in automotive manufacturing, a client analyst will inspect and review certain vehicles that come down an assembly line based on certain checkpoints. During these inspections, the analyst will mark down on a sheet of paper the type of defect that was detected using a certain color coding system. They will indicate where the defect was located and the size/severity. This product is geared to reduce the usage of paper as well as provide an automated way to generate the desired reports from any entered defects from the three GM plants: Lansing Delta Township, Lansing Grand River, and Lake Orion Assembly.

This product will allow the analyst to log into a system for their plant. The analyst will select which plant they are located at, select which position of the car the defect is located at (roof, left, right), and select which vehicle model to use. The analyst will then select where the defect is located on a grid of the model. This is done by selecting the row and column that most accurately identifies the location of the defect. The analyst will then select the severity (Sev I - Sev V or Special) and size (small, medium, large) of the defect.

This product will access a database that will automatically process and collate requested reports defined by the user. This database will be shared among the three different GM plants to provide data and insight for defect analysis.

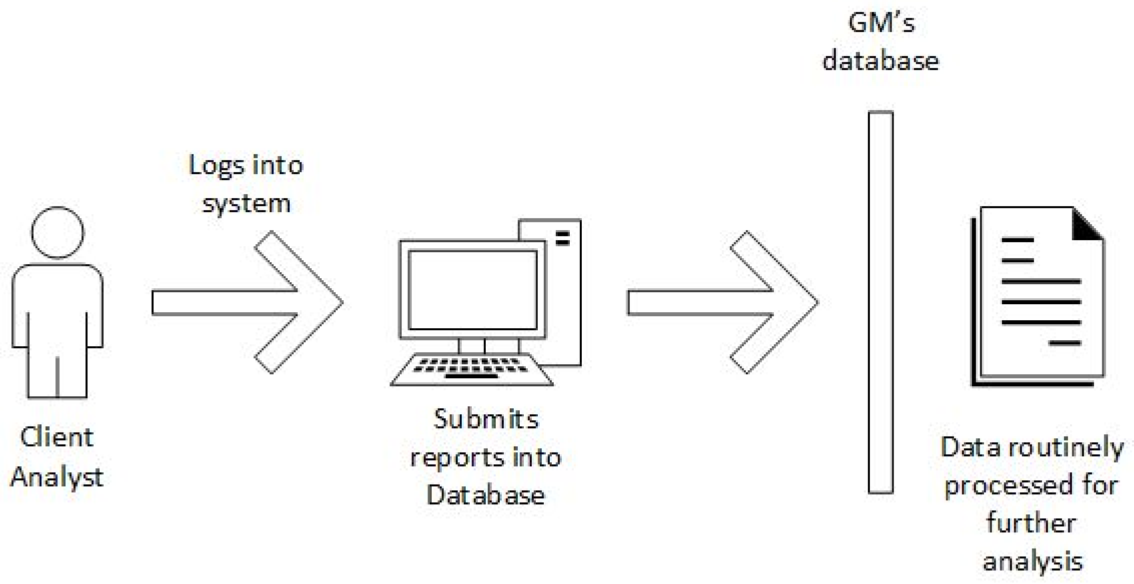

GM's database
Logs into system
Client Analyst
Submits reports into Database
Data routinely processed for further analysis

Figure 2.1: System Overview

Due to the system being web-based, there will be an extra level of authentication that will verify that the user is accessing the website from a trusted domain. This means that users will be unable to access offsite. In addition, this application is not designed for mobile use, and therefore needs either a desktop computer or laptop to utilize the application.

Constraints of the system are that the software itself is not customizable, the user must use the system as designed. There also may be memory constraints as the system will permanently store all defects that are submitted. However, this constraint can be combatted by consistently updating the memory space on the server.

2.2 Product Functions

The application *GM Report System* consists of four major functions: an interface that allows the user to enter and submit reports of defects, a function to retrieve individual reports, a function to retrieve a specified range of reports that will automatically produce desired analysis of those reports, and lastly a function that will parse through the individual reports to provide the user with their requested information.

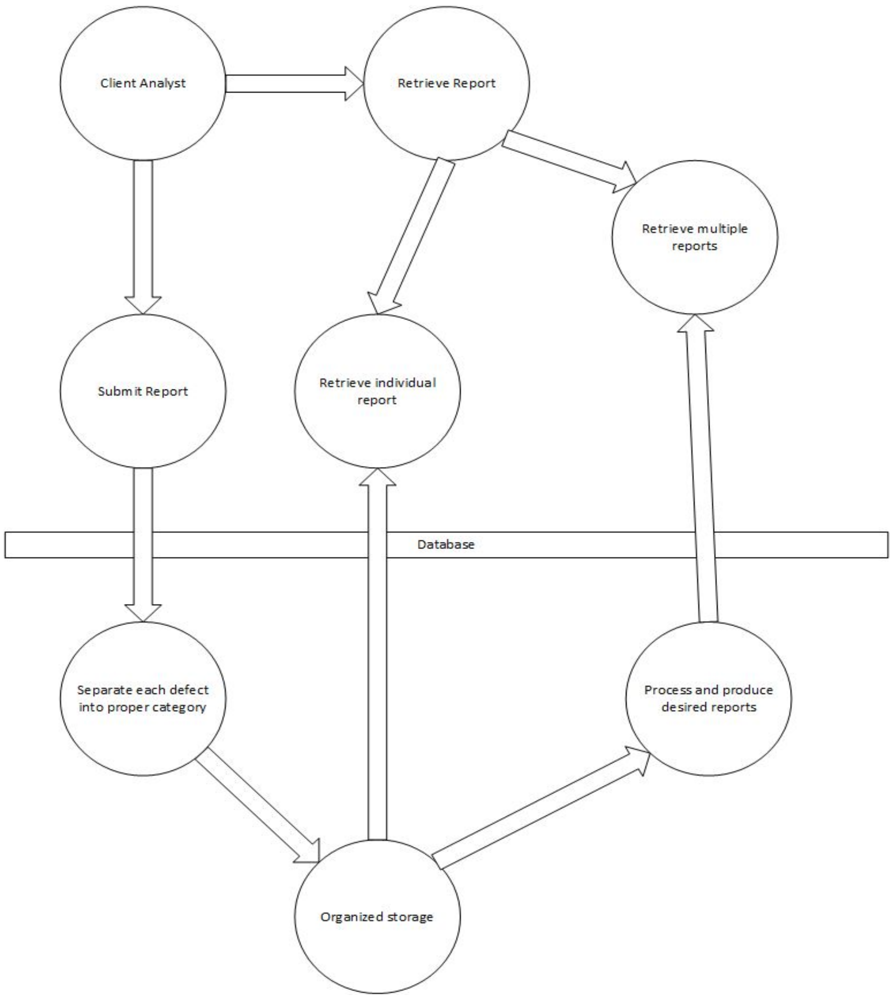

Client Analyst
Retrieve Report
Retrieve multiple reports
Submit Report
Retrieve individual report
Database
Separate each defect into proper category
Process and produce desired reports
Organized storage

Figure 2.2: Database/Client Interactions

2.3 User Characteristics

The expectations that the user should have in order to correctly use this product includes a very deep understanding of the different types of defects that occur on vehicles. The user will also need to understand the information that is produced from the product. Aside from needing certain credentials to log into the system, the product will be user-friendly enough to simply need a very basic understanding of computers.

2.4 Constraints

One constraint with this product is ensuring that there is always enough space within the database that contains the reports. This will require monitoring the system which will notify developers when space is almost full. Another constraint is not being able to verify if the user enters the intended information into the system. It’s possible to provide a pop-up that warns the user to double-check what they entered, but again it’s very likely that they could just skip this warning without checking. An additional constraint is that the user may not be using an updated system when using the application which could result in security risks within the application. A potential constraint of the software is precisely representing a defect on a digital system as the analyst will no longer be able to draw the defect and must make judgements on size and severity which have restricted options on the system.

2.5 Assumptions and Dependencies

Due to the fact that the product is a web-based system, we assume that users will have a hardware system that is capable of accessing the internet over a secured network. We also assume that the user's system is always updated with the latest version of the browser that they will use to operate the product. For security, the user will only access the product at any one of the three GM plants. We assume that GM will handle this by requiring an extra layer of security to prevent access to the system from outside of the plant.

2.6 Apportioning of Requirements

Currently, the analyst is responsible for analyzing the data provided by the product and determining when a specific defect is occurring too often along with other similar issues. Future versions/releases of this product will provide assistance with notifying the user when it has noticed a pattern or a common occurrence of a specific defect.

3 Specific Requirements

1. User Authentication
   1.1. User (Client Analyst) must supply proper credentials to enter the system.
2. Defect Analysis & Data Entry
   2.1. Report must contain the following user/plant information:
      2.1.1. User id
      2.1.2. Checkpoint
      2.1.3. Plant name
   2.2. Report must contain a minimum of 1 defect.
   2.3. Report must not contain more than 100 defects.
   2.4. Report must contain defect location on vehicle.
      2.4.1. Top side
      2.4.2. Left side
      2.4.3. Right Side
   2.5. Report must contain the car model from a list of available models.
   2.6. Report must contain the size/severity of the defect on the car.
      2.6.1. Small (Severity 1)
      2.6.2. Medium (Severity 5)
      2.6.3. Large (Severity 10)
   2.7. Defects contained in report must be displayed on grid system.
3. Report Generation
   3.1. Reports must be generated from one of the following checkpoints
      3.1.1. Polish Deck
      3.1.2. Prime Review
      3.1.3. E-Coat
   3.2. Reports with a focus on the defects per unit must be generated by the system every day.
   3.3. Reports with a focus on defects per surface must be generated by the system once per week.

3.4. Reports with a focus on the trends must be generated by the system once per month.

3.5. Reports must be printed after generation.

3.6. After a report is generated it needs to be pushed to the database for persistent storage.

3.7. Reports displayed must be pulled from the database.

4. System Application

4.1. Software system must be capable of utilization by multiple GM plants.

4.1.1. Lansing Delta Township

4.1.2. Lansing Grand River

4.1.3. Lake Orion Assembly

4 Modeling Requirements

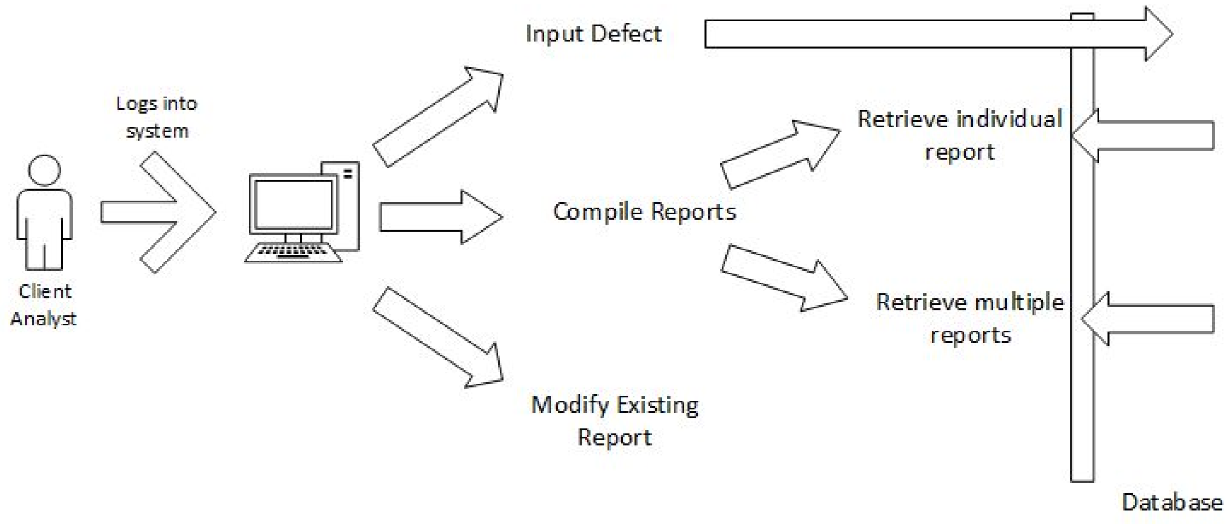


Figure 4.1 Overall system use case

Class Diagram

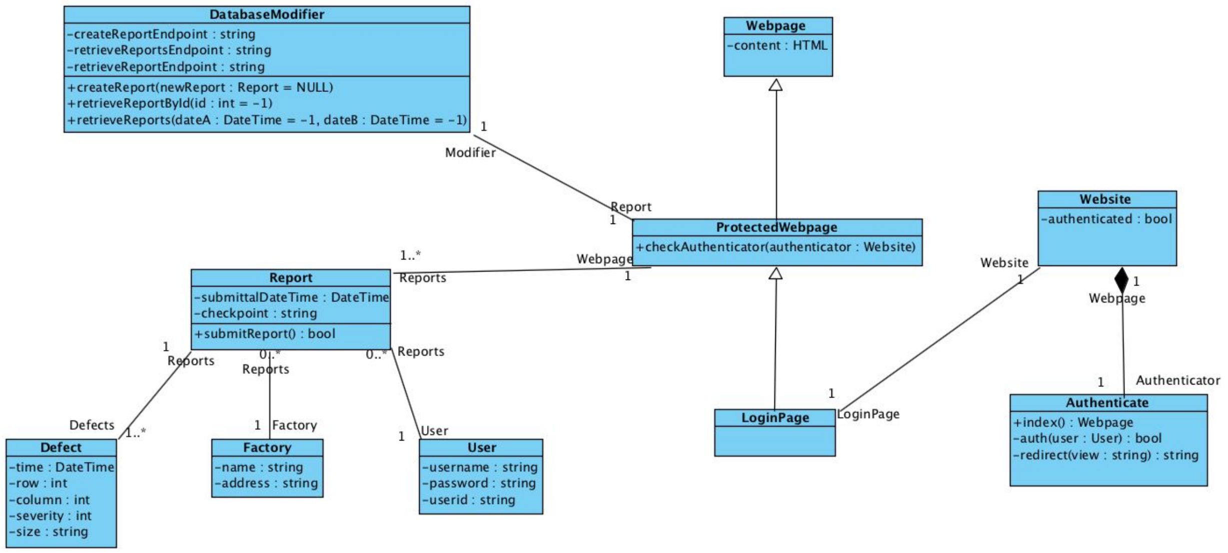


Figure 4.2: System Class Diagram (UML)

User Authentication

Use-Case Diagram

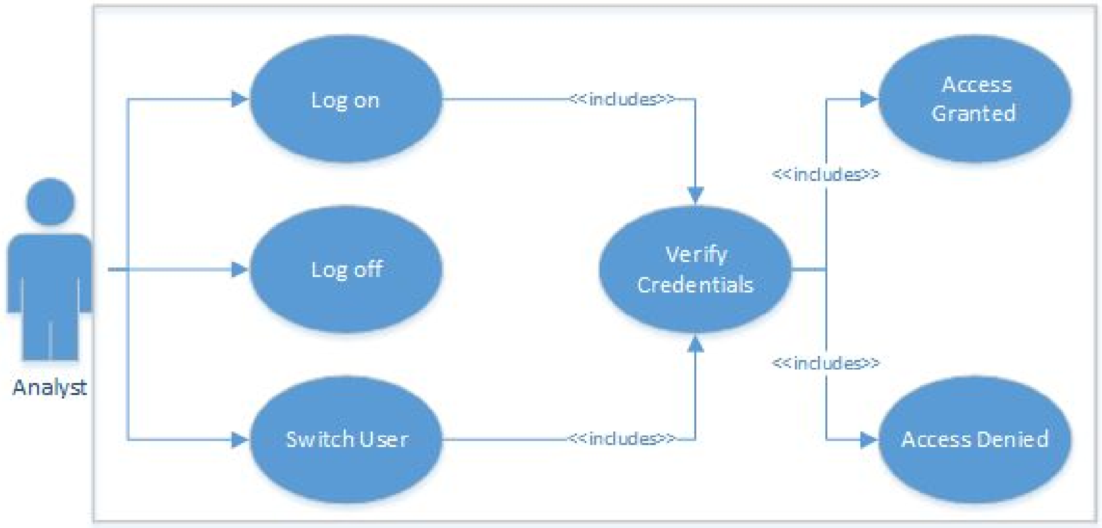


Figure 4.3: User Authentication Use-Case Diagram

Sequence Diagram

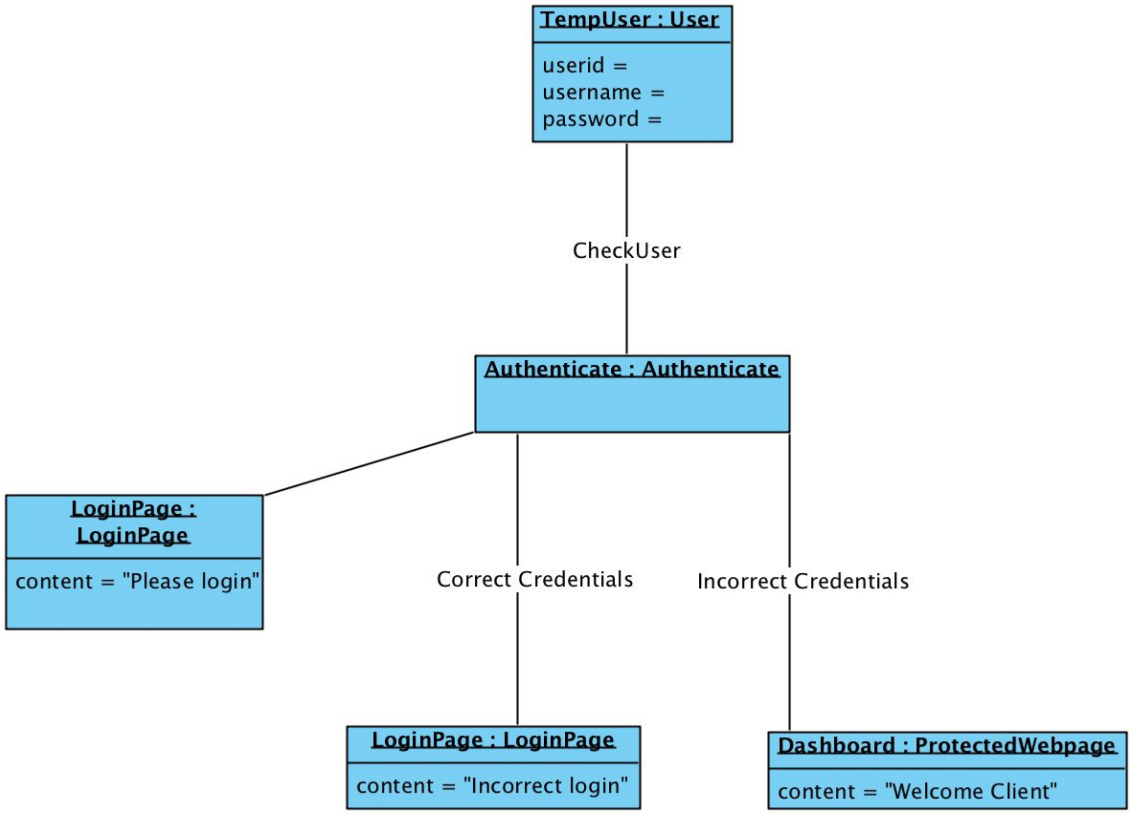


Figure 4.4 User Authentication Sequence Diagram

English Description

Scenario 1: Client A directs their browser to the application, navigates to the login page, inputs their credentials and clicks “Login”. The system detects that the credentials supplied were invalid and redirects Client A to the login in page with an error message: “Incorrect login”.

Scenario 2: Client A then double-checks their email and password, re-enters on their credentials in the login page and clicks “Login”. The system verifies that the credentials supplied were valid and logs Client A into the system and redirects them to the dashboard page.

State Diagram

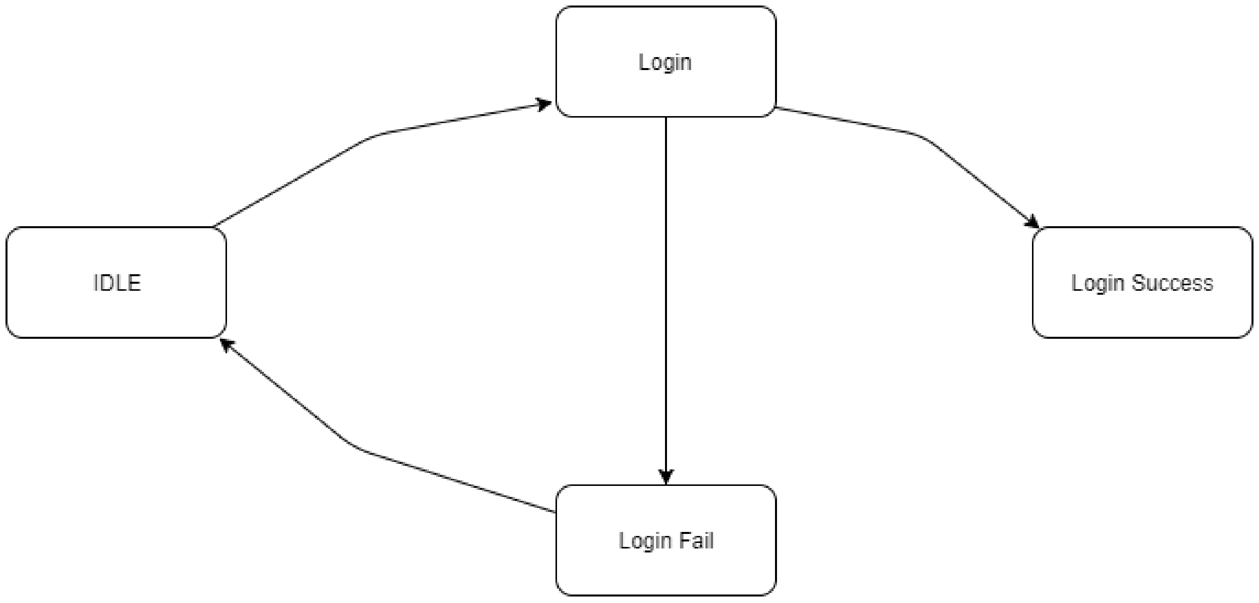


Figure 4.5: User Authentication State Diagram

Defect Analysis and Data Entry

Use-Case Diagram

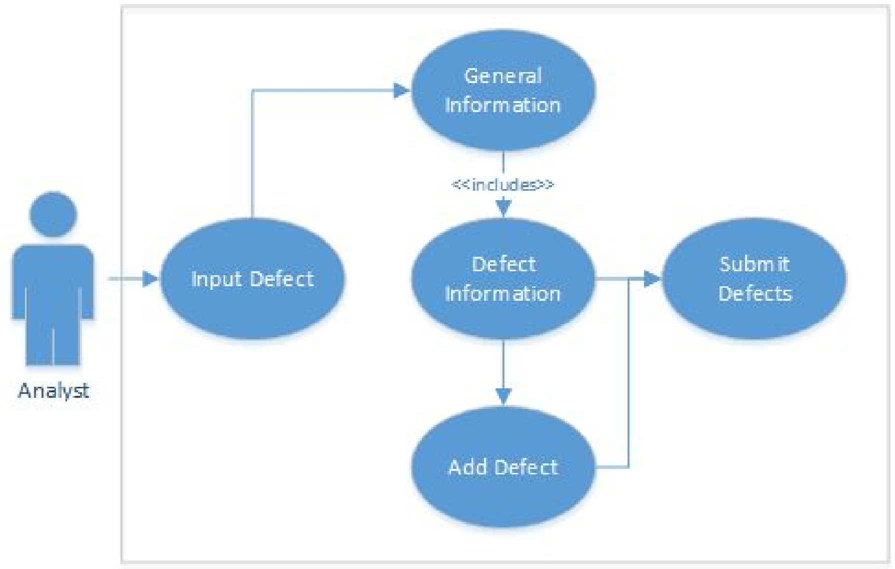


Figure 4.6: Defect Analysis Use-Case Diagram

Sequence Diagram

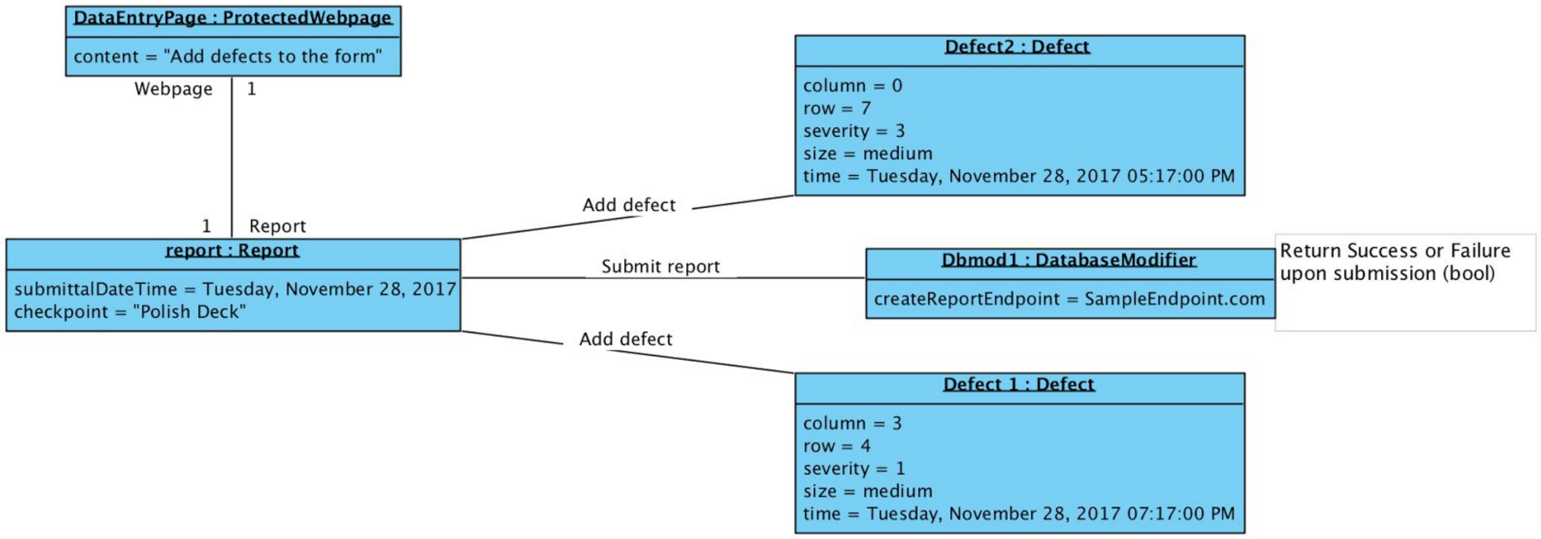

DataEntryPage : ProtectedWebpage
content = "Add defects to the form"
Webpage
1
1
Report
report : Report
submittalDateTime = Tuesday, November 28, 2017
checkpoint = "Polish Deck"
Add defect
Defect2 : Defect
column = 0
row = 7
severity = 3
size = medium
time = Tuesday, November 28, 2017 05:17:00 PM
Submit report
Dbmod1 : DatabaseModifier
createReportEndpoint = SampleEndpoint.com
Return Success or Failure upon submission (bool)
Add defect
Defect 1 : Defect
column = 3
row = 4
severity = 1
size = medium
time = Tuesday, November 28, 2017 07:17:00 PM

Figure 4.7: Defect Analysis Sequence Diagram

English Description

Scenario 1: After Client B has logged in, they go to the data entry page, select the plant name, report type, model, and position. They will then input individual defects into the system and click “Submit”. The system is currently unavailable, and rejects the report submission and alerts Client B that the submission has failed. All the input data on the form will stay intact.

Scenario 2: Client B waits a few seconds and clicks “Submit” again. The form is serialized and successfully submitted to the system and the system alerts Client B that the submission has succeeded and redirects the user to a new report form.

State diagram

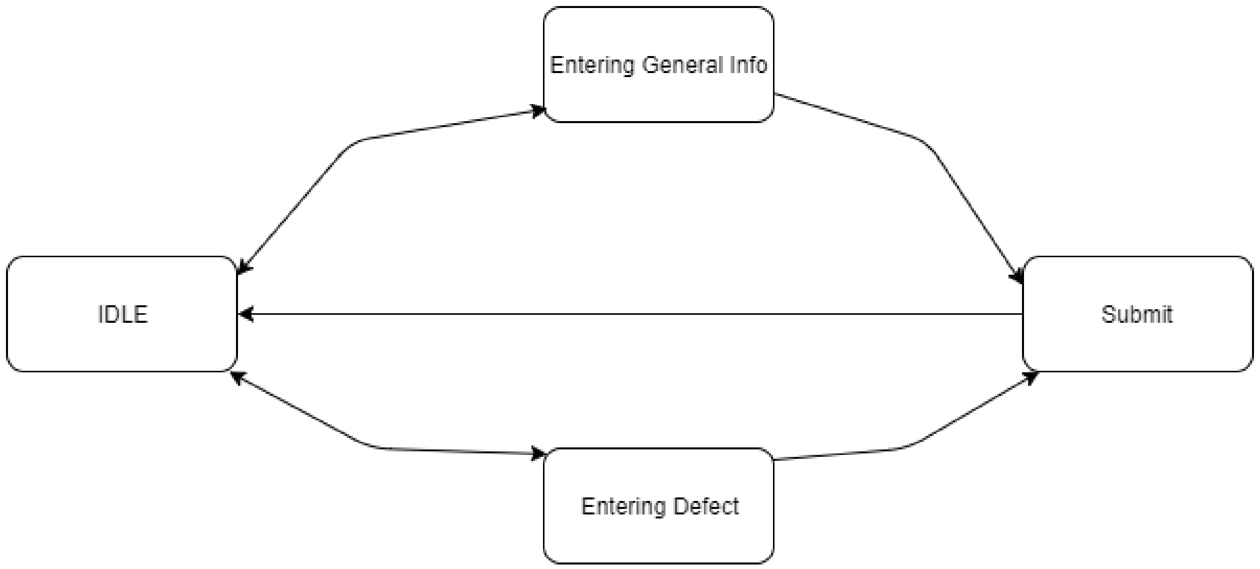


Figure 4.8: Defect Analysis State Diagram

Report Generation

Use-Case Diagram

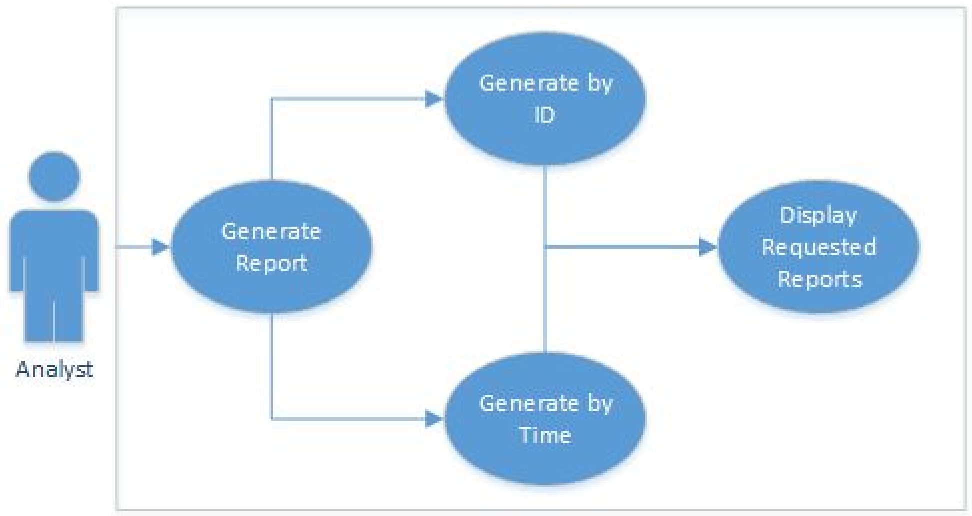


Figure 4.9: Report Generation Use-Case Diagram

Sequence Diagram

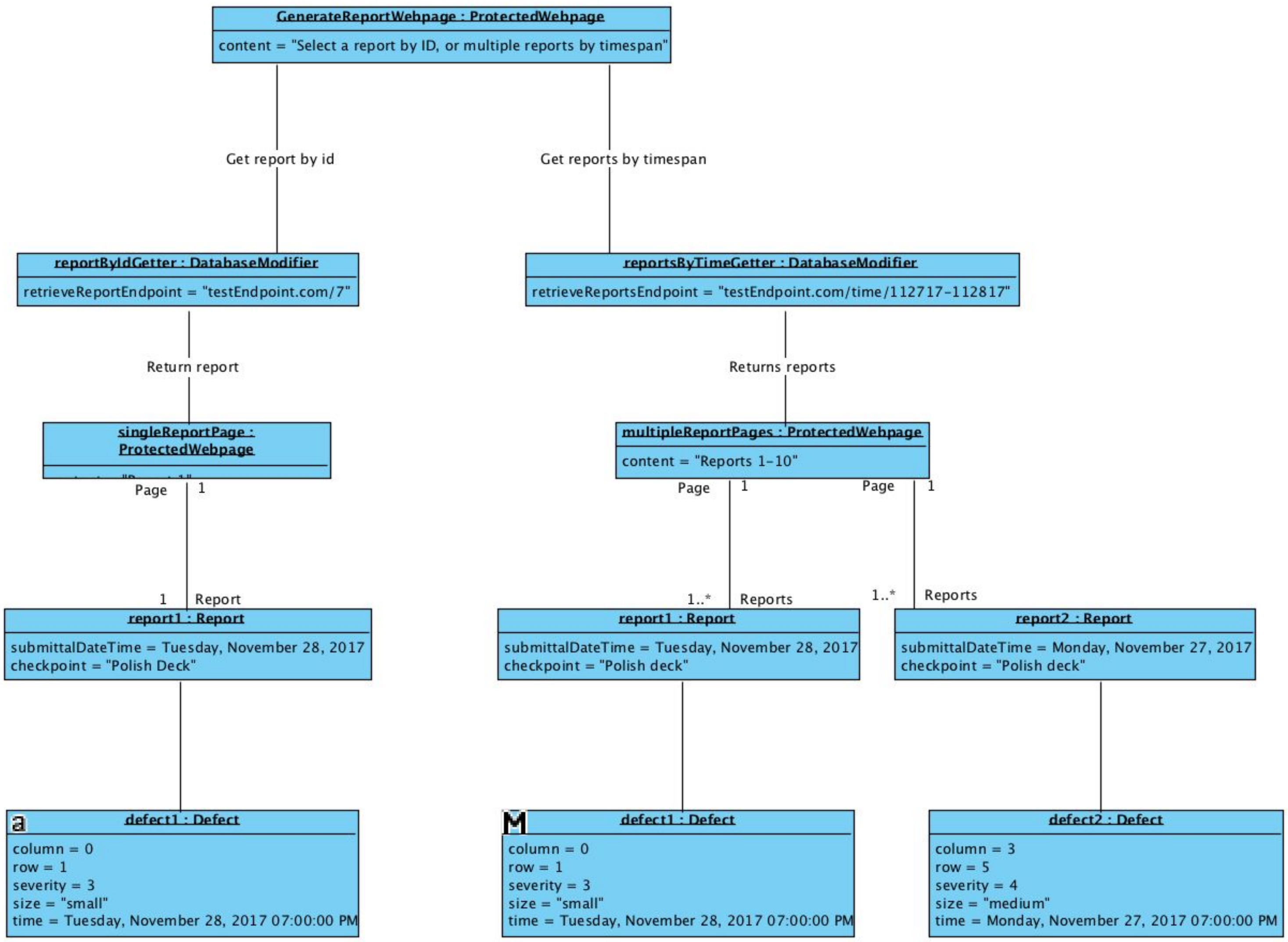

GenerateReportWebpage : ProtectedWebpage
content = "Select a report by ID, or multiple reports by timespan"
Get report by id
Get reports by timespan
reportByIdGetter : DatabaseModifier
retrieveReportEndpoint = "testEndpoint.com/7"
reportsByTimeGetter : DatabaseModifier
retrieveReportsEndpoint = "testEndpoint.com/time/112717-112817"
Return report
Returns reports
singleReportPage : ProtectedWebpage
multipleReportPages : ProtectedWebpage
content = "Reports 1-10"
Page
1
1
Report
Reports
1..*
report1 : Report
submittalDateTime = Tuesday, November 28, 2017
checkpoint = "Polish Deck"
report1 : Report
submittalDateTime = Tuesday, November 28, 2017
checkpoint = "Polish deck"
report2 : Report
submittalDateTime = Monday, November 27, 2017
checkpoint = "Polish deck"
defect1 : Defect
column = 0
row = 1
severity = 3
size = "small"
time = Tuesday, November 28, 2017 07:00:00 PM
defect2 : Defect
column = 3
row = 5
severity = 4
size = "medium"
time = Monday, November 27, 2017 07:00:00 PM

Figure 4.10: Report Generation Sequence Diagram

English Description

Scenario 1: Client C has successfully input numerous reports into the system and they would like to see a specific report, so they go to the generate report page and click on a specific report id. The system then compiles the specific report and presents it to the user.

Scenario 2: Client C would also like to see a compiled report within a specific timespan and they choose the start and end time on the generate report page and clicked “Submit”. The system compiles all the reports within the timespan into one and presents it to Client C.

State diagram

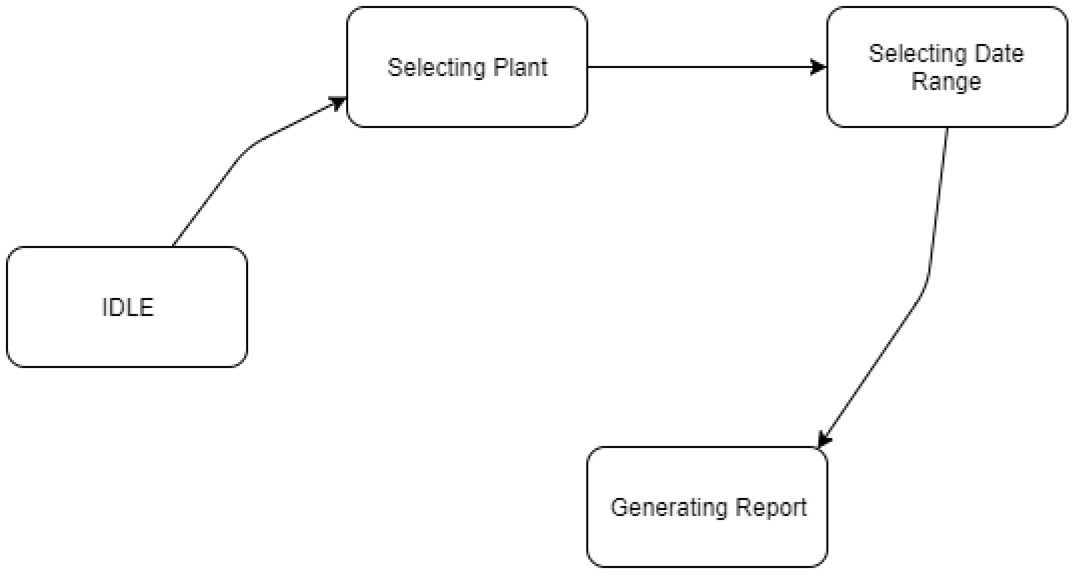


Figure 4.11: Report Generation State Diagram

System Application

Use-Case Diagram

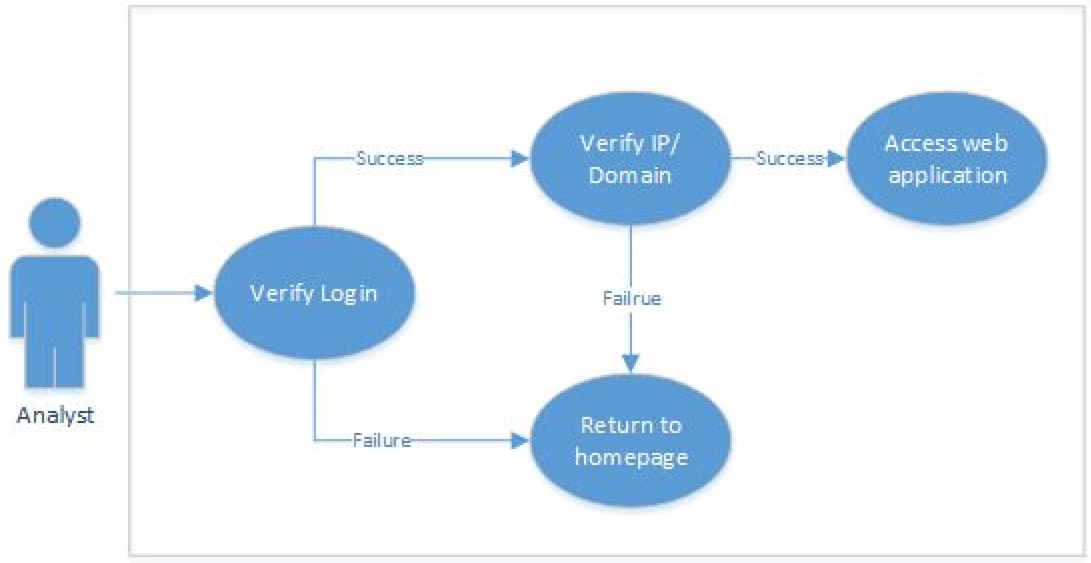


Figure 4.12: System Application Use-Case Diagram

Sequence Diagram

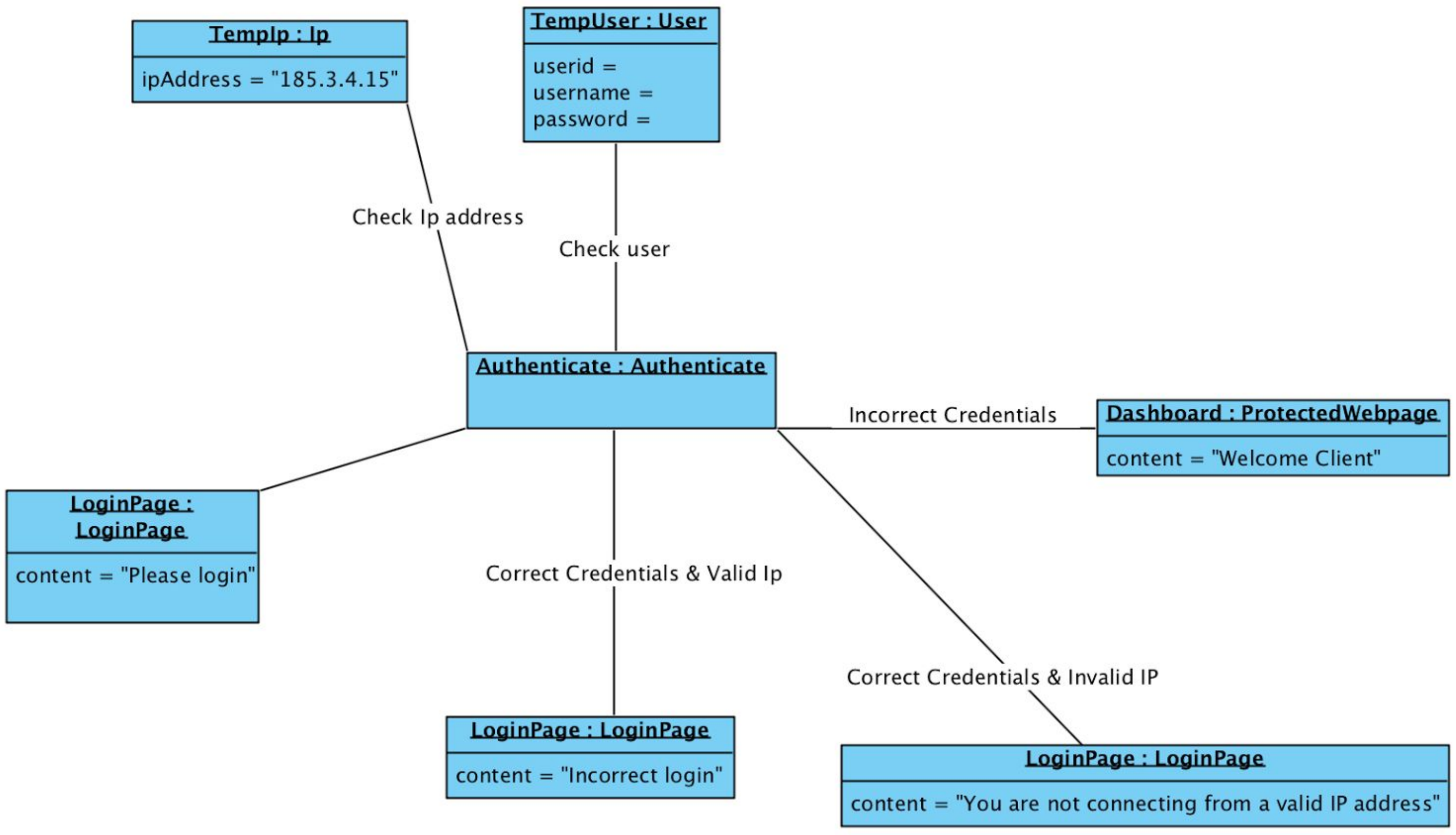

TempIp : Ip
ipAddress = "185.3.4.15"
TempUser : User
userid =
username =
password =
Check Ip address
Check user
Authenticate : Authenticate
Incorrect Credentials
Dashboard : ProtectedWebpage
content = "Welcome Client"
LoginPage : LoginPage
content = "Please login"
Correct Credentials & Valid Ip
Correct Credentials & Invalid IP
LoginPage : LoginPage
content = "Incorrect login"
LoginPage : LoginPage
content = "You are not connecting from a valid IP address"

Figure 4.13: System Application Sequence Diagram

English Description

Scenario 1: Client D from one of the plants logs into the system. The system verifies that the client's IP address is from one of the plants, redirects the user to the application.

Scenario 2: Malicious user E acquired Client D's credentials, logs in on a public network. The system notices that the IP address is not from one of the authorized plant, fails the login and redirects the user to homepage.

State diagram

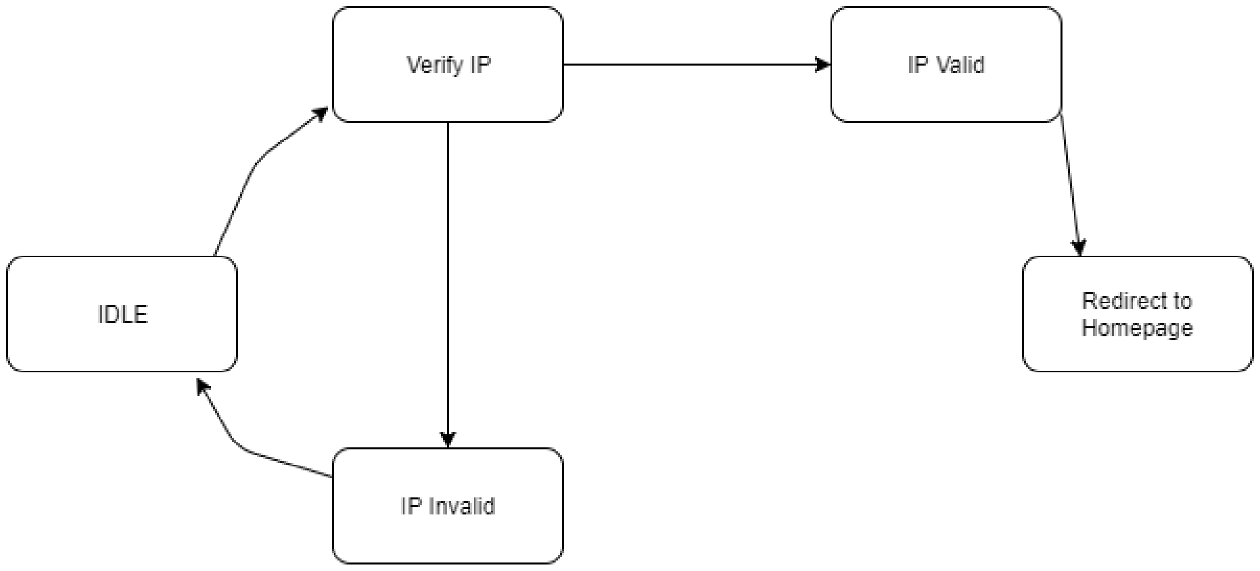


Figure 4.14: System Application State Diagram

Use Case Tables

Use Case Name:	Input Defect
Actors:	Client Analyst
Description:	Client will submit a report consisting of 1 or more detected defects. The report will be separated individually into the database to allow easier collation when requested by the client analyst.
Type:	function

Use Case Name:	Compile Reports
Actors:	Client Analyst
Description:	When prompted by the client analyst, the system will retrieve the requested amount of reports. If the analyst chooses only one report, he will be given all the necessary information for that specific report. If the analyst chooses to analyze multiple defects,, the product will gather the requested range of defects and compile it into the desired report.
Type:	function

Use Case Name:	Modify Existing Report
Actors:	Client Analyst
Description:	Using general information about the defect, the client analyst can retrieve a specific defect to either modify or delete. If choosing to delete, the analyst must explain why before finishing. Afterwards, the defect will stay within the system for 3 months before being completely removed, but will not be included in any future reports.
Type:	function

Class Tables

Element Name		**Description**
Factory		A specific GM plant
Attributes		
	Name: string	Name of the plant
	Location: string	Location of the plant

Element Name		**Description**
Defect		The defect for a car
Attributes		
	Type	Type of defect
	Location	Location of defect on car
	Severity	Severity of the defect
	Size	The defect size
	Time	Date and time of defect recorded
	User	User id who submitted report

Element Name		Description
Report		Type of report
Attributes		
	Date	Date of the report
	Shift	Shift of the analyst

Element Name		Description
DatabaseModifier		Class to interact with the database
Attributes		
	createReportEndpointURL	Endpoint url for creating a defect report
	retrieveReportsURL	Endpoint url for retrieving a group of reports
	retrieveReportURL	Endpoint url for retrieving a single report
Methods		
	createReport	Create a report
	retrieveReportById	Retrieve a single report
	retrieveReports	Retrieve a group of reports

Element Name		**Description**
Webpage		Class for web page representation
Attributes		
	content	The HTML content of the web page

Element Name		**Description**
ProtectedWebpage: Webpage		Class for the web application
Attributes		
	content	The HTML content of the web page
Methods		
	checkAuthenticator	Check whether the user has logged in

Element Name		**Description**
Website		Class to interact with the database
Attributes		
	authenticated	A boolean for user authentication status

Element Name		Description
Authenticate		Class for authenticating user
Methods		
	index	Return the index web page
	auth	Authenticate a user’s credentials
	redirect	Redirect a user to a certain web page

Element Name		Description
User		Class for the user representations
Attributes		
	username	The user’s username
	password	The user’s password
	userid	The user’s unique id

5 Prototype

The prototype will be provided as a web interface, there is no external dependencies for it to run. The system will be designed as a two-part application: the web interface, and the Python coded backend. The web interface will utilize PHP, JavaScript, HTML and CSS as its building block, where the backend will use Python and a database engine (possibly

SQLite/GDBM). The interface will send the requests to the backend, and the backend shall process the input, serialize, persist and generate reports based on persisted dataset.

There will be no plugins except for the required libraries at each end. For the frontend, we will be needing Axios javascript library for initiating HTTP requests; for the backend, we will be needing database drivers for persisting and querying datasets, Falcon (2) and Gunicorn (3) for REST interface.

The frontend interface will provide the clients with a form to file, and the data collected will be sent to the backend server, the backend server then can persist the data in a database. Depending on the request, the backend server will retrieve stored data and generate reports either periodically or on-demand, and send the compiled data to the web interface for presentation.

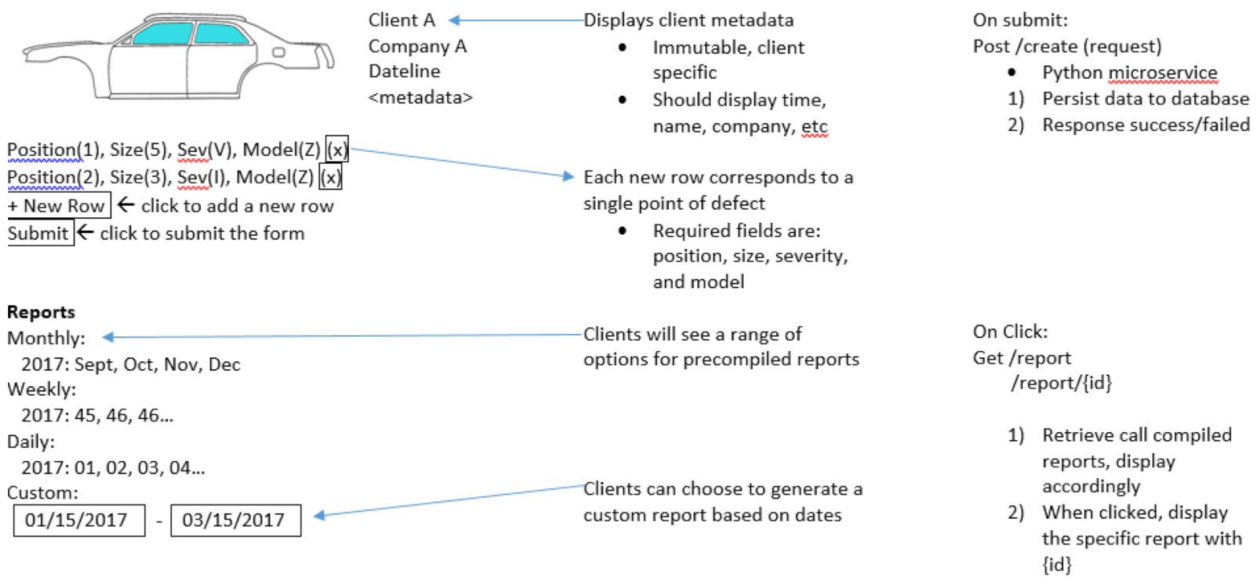
5.1 How to Run Prototype

In order to run the prototype, the service provider will need two sets of programs. For the frontend, which was built on-top of PHP (Laravel), SQLite, VueJS, Axios, etc., the service provider needs a host system that at least has PHP interpreter, HTTP server, NodeJS. For this task, the work group has chosen to run the application on a FreeBSD (4) host with PHP version 7.1.11, Nginx (5) version 1.12.2 and NodeJS (6) version 8.9.1. To make the interface more secure, the work group has chosen Let's Encrypt (7) as its SSL certificate vendor for HTTPS support.

For the backend, the service provider will need the latest Python 3 environment. Since the Python platform bundles with SQLite and GDBM support, the group decided to use these libraries as the persistent database store for the reported data. In order to interface with the frontend, the backend is required to provide a REST API, and the group chose the Falcon library and Gunicorn library for this task.

The clients, only need the latest browser to use the prototype, simply direct the browser to https://tmp.ink/prototype, login with a valid account, and then it is ready to use. The web application is designed to fit for multiple platforms, this flexibility allows the client to run the application on a cell phone, a tablet, a laptop or a desktop.

5.2 Sample Scenarios



Client A
Company A
Dateline
<metadata>

Displays client metadata
- Immutable, client specific
- Should display time, name, company, etc

On submit:
Post /create (request)
- Python microservice

1) Persist data to database
2) Response success/failed

Position(1), Size(5), Sev(V), Model(Z) (x)
Position(2), Size(3), Sev(I), Model(Z) (x)
+ New Row ← click to add a new row
Submit ← click to submit the form

Each new row corresponds to a single point of defect
- Required fields are: position, size, severity, and model

Reports
Monthly:
2017: Sept, Oct, Nov, Dec
Weekly:
2017: 45, 46, 46…
Daily:
2017: 01, 02, 03, 04…
Custom:
01/15/2017 - 03/15/2017

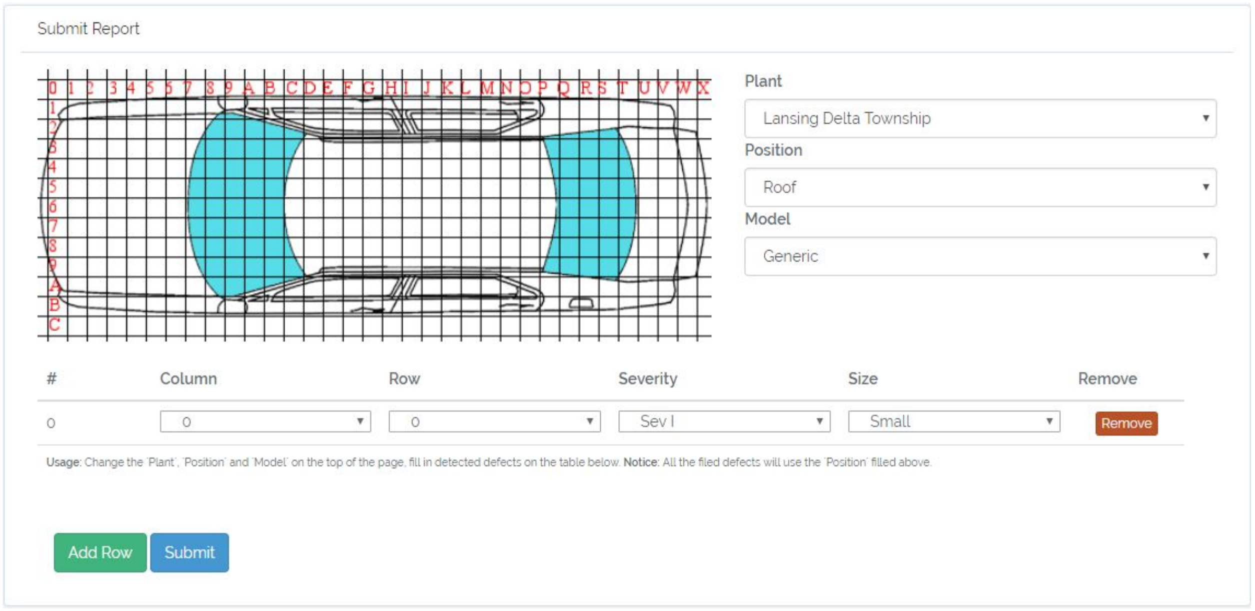
Clients will see a range of options for precompiled reports

Clients can choose to generate a custom report based on dates

On Click:
Get /report
/report/{id}

1) Retrieve call compiled reports, display accordingly
2) When clicked, display the specific report with {id}

Figure 5.1: Prototype example 1


Submit Report

0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X

Plant: Lansing Delta Township

Position: Roof

Model: Generic

#	Column	Row	Severity	Size	Remove
0	0	0	Sev I	Small	Remove

Usage: Change the 'Plant', 'Position' and 'Model' on the top of the page, fill in detected defects on the table below. **Notice:** All the filed defects will use the 'Position' filled above.

Figure 5.2: Prototype Interface

English Description of Prototype Example

Client A, directs their browser to the application, and completes the form above. Each row contains these fields: position, size, severity and model, in this example, the user filled three rows:

position (1), size (5), severity (IV), model (Z) [X]

position (2), size (4), severity (III), model (Z) [X]

position (3), size (2), severity (I), model (Y) [X]

After entering the data, Client A submits the form. The backend then handles the submitted form, serializes the data and saves it in the database. The saved metadata contains: user_id, company_id, created_at and report_id, then for each row, a record is created that contains the position, size, severity and model, the data will then be permanently saved within the database.

If Client A navigates to the report section, they will see pre-compiled reports grouped by months, weeks, and days, if the user clicks any one of the reports, the browser will show the compiled report in a new window. A client is given a choice to generate a custom report based on timeline, once the client selected desired time span, the backend server will generate a new report and directs the client to this report.

6 References

1. D. Thakore and S. Biswas, "Routing with Persistent Link Modeling in Intermittently Connected Wireless Networks," Proceedings of IEEE Military Communication, Atlantic City, October 2005.
2. Falcon Contributors. "The Falcon Web Framework" https://falcon.readthedocs.io/en/stable
3. Gunicorn Contributors. Gunicorn - Python WSGI HTTP Server for UNIX" http://gunicorn.org
4. The FreeBSD Foundation. "The FreeBSD Project" https://www.freebsd.org
5. NGINX Inc. "NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy" https://www.nginx.org
6. Node.js Foundation. "Node.js" https://www.nodejs.org
7. Internet Security Research Group. "Let's Encrypt - Free Certificates" https://letsencrypt.org/
8. Study Group 4. "Study Group 4", https://tmp.ink
9. Software Engineering CSE435. "Automotive Paint Defect Analysis." 10 October 2017, www.cse.msu.edu/~cse435/Projects/F2017/Project-Description.pdf

7 Point of Contact

For further information regarding this document and project, please contact **Prof. James Daly** at Michigan State University (daly at cse.msu.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.