Laboratory Report

SOFTWARE ENGINEERING AND TESTING METHODOLOGIES CSE2017L

SCHOOL OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted By	
Student Names	Swarup Ghosh
Enrolment Number	170020203040
Section/Group	A-T2
Department	Computer Science and Engineering
Session/Semester	2018-19/ Even Semester
Submitted To	
Faculty Name	Ms. Sujata
	Ms. Vaishali Arya

G D GOENKA UNIVERSITY

GURGAON, HARYANA

Software Requirements Specification

Online Shopping Management System
v1.0

CONTENTS

Objectives

The primary objective of the project is to provide customers (users) an efficient interface that can allow them to buy products through an e-commerce application. The online shopping management system is intended to provide the customers different products and services which they can buy online which will try to replace the standard brick and mortar shopping experience in place. It will primarily assist users with a hassle free and seamless shopping experience.

Apart from that there are a few secondary objectives that our project will fulfil, which have been enlisted underneath.

- To provide an effective catalogue of products to each individual user
- To harness use of recommender systems in order to suggest products to existing users on the platform
- To enable user engagement at large through our system, promotion and discount related services are to be incorporated
- To deliver a highly available system that would be designed with the needs of the user in mind
- To have an in-built mechanism that acts as an incident management system, where in people maintaining the software keep a record of the failures that occur in the system
- To incorporate reporting tools in the platform such that people working in the sales and marketing teams can submit generic reports about products or a category of products

Requirements

Functional Requirements

The online shopping management system constitutes different modules in the form of web based micro-services that actually interact amongst each other to serve a highly available REST API (Representational State Transfer - Application Programmers Interface). Use of micro services internally ensures the fact that if in case of failure of any sub system, the entire system won't be affected at large since other sub systems will be working with the same availability rate.

The REST API is used further by client applications like mobile and web apps which is functioning at the user end.

Module 1: JSON REST API

Each sub module is implemented in the form of a web micro service.

Module 1.1: Catalogue

The catalogue provides **users** with details about each **product** that is available for purchase. Apart from that it lists all products on the basis of its **product category**.

Constraints: The item catalogue is like a nested data structure due to product categorisation which may sometimes be more convenient to be persisted on NoSQL databases rather than chosen SQL databases.

Module 1.2: Discount and Promotion

The system allows **discounts** and special **vouchers** by use of referral codes and **promotion** oriented **discounts** on individual subset of **products**. **Discount** to be provided across product categories for products whose sales is not upto the seller's expectation.

Constraints: Voucher codes are not forecasted automatically to external online platforms which makes manual intervention necessary at administrators end whenever a voucher code is generated.

Module 1.3: Cart and Checkout

For any user purchase of items serves as the most important functionality. **Cart** is the virtual place which allows users to persist products selected by user for purchase even across **sessions**. Whereas the **checkout** functionality actually redirects to the payment page so that an **order** can be initiated with a callback mechanism.

Constraints: There may be issues relating to payments in case transactions are dropped by bank since the payment gateway is integrated on the platform through third party solutions.

Module 1.4: User Management and IAM

For any online system the **users** module serves as a gateway for access control to the the outside world. The same module presents upon to its **users** with the functionality for **signup** and **login**. The identity access management based system help distinguish **standard users** and **administrators** on the platform. An OAuth based authentication mechanism takes care of

Constraints: A common security level is used for both administrators as well as standard users.

Module 1.5: Search Engine Optimisation Dashboard

A dashboard for SEO allows input of **metadata** and creation of search engine indexes optimised for web **crawlers** and search engine detections. It includes automatic **sitemap** generation.

Constraints: Regular generation of sitemaps manually by the administrator is advised failing to do which would result in incorrect indexing of site at the search engine end.

Module 1.6: Emailer

Both **transactional emails** as well **marketing campaigns** forms an integral module for any e-commerce solution. This module targets **users**, sending them emails for details about orders as well as for product promotions.

Constraints: Multiple emails being sent to the same user for product promotion purposes may result in emails being classified as spam at the user end due to strong spam classifiers.

Module 1.7: Reporting Tools

This module is useful for **administrators** to generate meaningful reports about **products** and **product categories** based on data that has been already collected on the platform.

Constraints: Due to use of third party services for courier facility, cash on delivery services might not be available for all regions but it will indirect ensure a country-wide deliverability of goods.

Module 1.8: Support Tickets

During functioning of the system it is obvious for the **users** to face issues and problems. A s**upport ticket** mechanism acts as a bridge between **administrators** and **users** to mediate such issues through communication.

Constraints: Administrators may not be available for support resolution at all times which may result in delays at user end.

Module 1.9: Order Book and Returns

Order books are to be maintained per **user** which records all **transactions** as well as allows initiation of item **returns**.

Constraints: Returning of items is available for each product which may not be desirable at all times by the seller.

Module 1.10: User Product Recommender

This module is used to recommend products through a recommender system based on the data collected from user activity.

Constraints: Product recommendations may be redundant in some cases, since it is based on user activity exclusively instead of data from third party advertisement services.

Module 2: Android Application

Module 2.1: Models

Models are constructed as a part of the application to facilitate interaction with the different modules present in the REST API. A single model and relevant classes are created for each module.

Module 2.2: Viewer (UI)

The application when being used from a mobile app frontend includes UI layouts which call different models to run the actual BL so as to provide users with a workable interface.

Module 3: Web Application

Module 3.1: Models

Models similar to the mobile application are created for the web application as well.

Module 3.2: Viewer (UI)

The viewer contains markups so to display different layouts and render web pages on the basis of the constructed models. It is similar to the mobile application viewer.

Module 3.3: Administration Dashboard

An administration dashboard available exclusively from the web application ensures that administrator is able to access necessary tools for the system management. Internally, it makes API calls for the dashboard functionalities.

Non-Functional Requirements

A high level view of the system is as follows. The non-functional requirements for the proposed system mainly constitute the following:

- Load Balancer(s)
    - nginx or any cloud native LB
- Spanning Web Server(s)
    - Apache Tomcat or similar
- Database Server(s)
    - Oracle MySQL server or any cloud native database service
- Internal Network for Micro Services

The application will be written in a way to reduce vendor lock-in by drastically putting implementations over open technologies rather than proprietary services. Hence, the entire application may be deployed to any cloud application platform or alternatively to any on premise platform with little or no change in internal configuration.

The system is designed such that the mobile app, web frontend based application internally communicates with REST API to provide the different functionalities to the user.

Security Requirements

The following security requirements are to be considered for the application:

- Use of HTTPS (for encrypted end to end communication)

- Use of OAuth or similar authentication mechanism (for user authentication over API)

- Use of encrypted SMTP, SPF and DKIM (for sending emails securely and lowering of email spam rates)

Methodology

An **agile** approach would be extremely suitable for deliverability of such a software product which has multiple modules consisting of a broad set of features that will evolve over time. In conjunction with traditional agile methodologies, **DevOps** tools can be significantly harnessed in order to leverage software delivery through use of CI/CD pipelines which will allow code to be tested, evaluated and rolled to production with minimal efforts.

Tool Specification

The following tools are used in order to make the shopping system functional:

- PayTM Payment Gateway (payment gateway to accept payments from users)

- Any Continuous Integration Platform (DevOps tool reduces transition from development to operations time which is best suited for production environments)

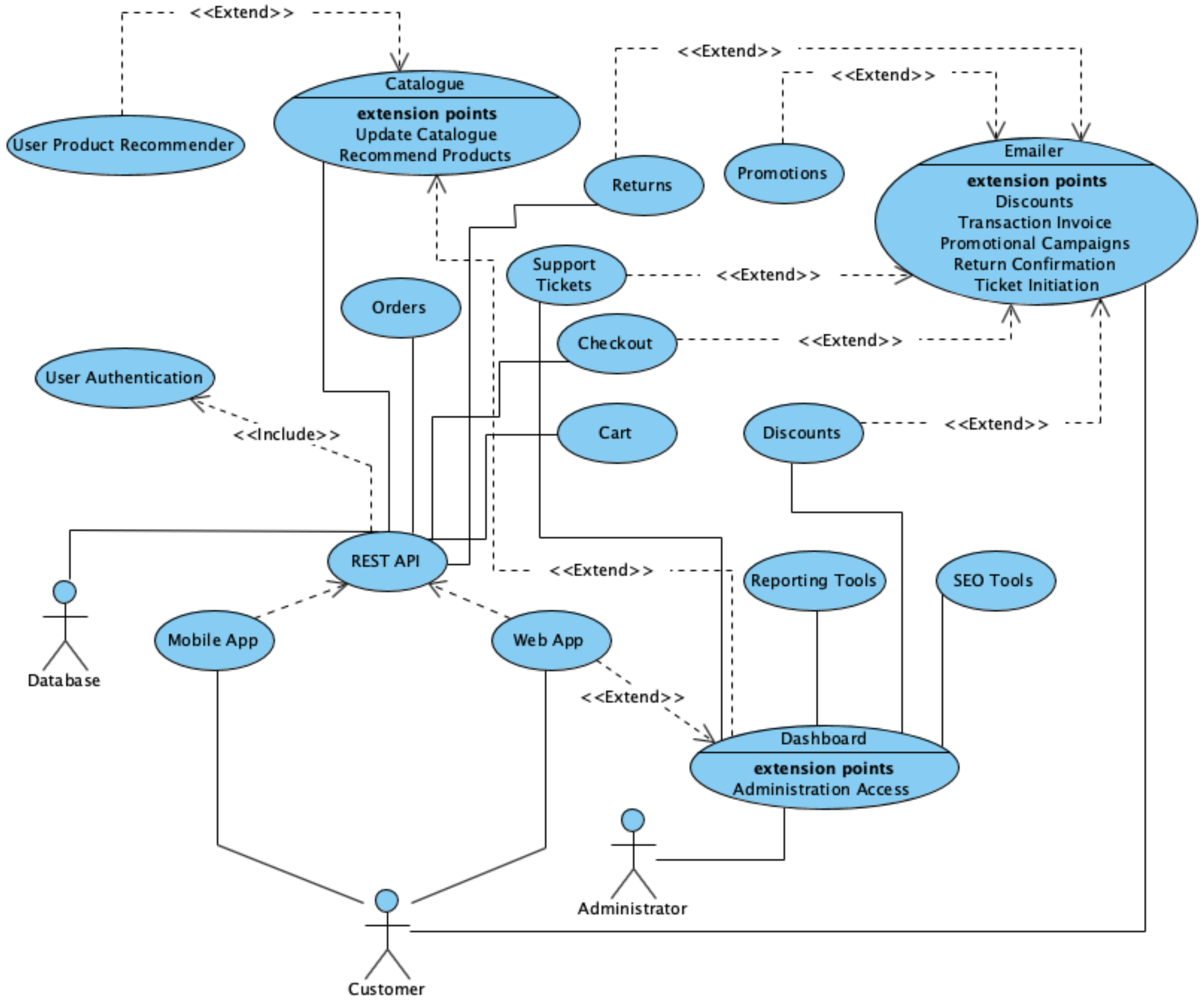
- Delhivery B2B Solutions (Delhivery courier services integration within the software platform for product shipping and returns)

- Any Cloud Platform or Micro Services Orchestration Platform (for cloud based deployments any suitable cloud platform may be used or alternatively micro service orchestration through Docker-Kubernetes like solutions in case of on-premise deployment)

Database Specification

A relational database management system will be used for the purpose of data storage and retrieval of the application. It will be deployed in such a way that it can be scaled and sharded for production workload environments. In this case, a standard Oracle mySQL database running InnoDB engine may be used or alternatively, MariaDB (an open source mySQL fork) may also be used since they allow interoperability.

Use Case Diagram



Overall Plan

The following tasks are to be performed chronologically during the development phase. The already mentioned iterative process development model for the project will ensure that each phase is properly tested and evaluated before delivery.

- The database will be created initially with the necessary schema that will be documented from the database design phase.

- The REST API will be developed that will establish database connections inside the application to facilitate the different functionalities.

- The android application models and web models will be prepared simultaneously.

- During the near end of the project the frontend design and mobile application layouts will be developed which will communicate with the respective model classes for working of the application.

It is notable to mention that during the development of REST API as a service a micro services architecture is to be followed. While during the web and mobile application are to conform with a simpler Model Viewer Controller (MVC) architecture.

Future Scope

The project will be aimed at high availability and ease of scalability such that it can be treated as a resilient system which allows to its users with shopping capabilities over online means replacing brick and mortar shops into digital carts. The future goal of the project would be to create a framework like application such that it can be deployed over the fly onto any application platform like a CMS (content management system) module. Thus, it will abstract away the internal working of the application providing a simple handy tool that is accessible directly to e-commerce administrators for making online shopping possible at user level.

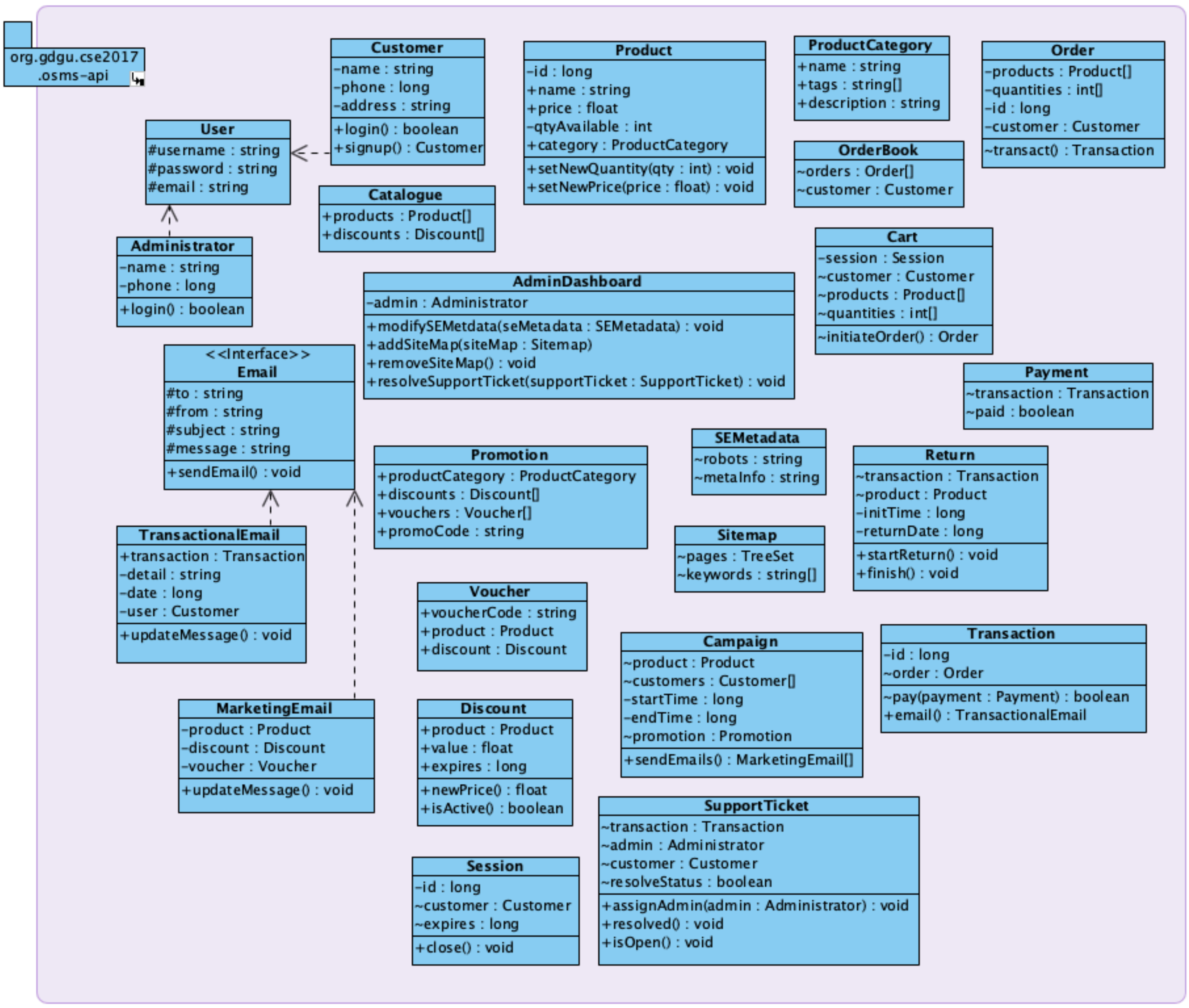
Design Document

Online Shopping Management System
v1.0

CONTENTS

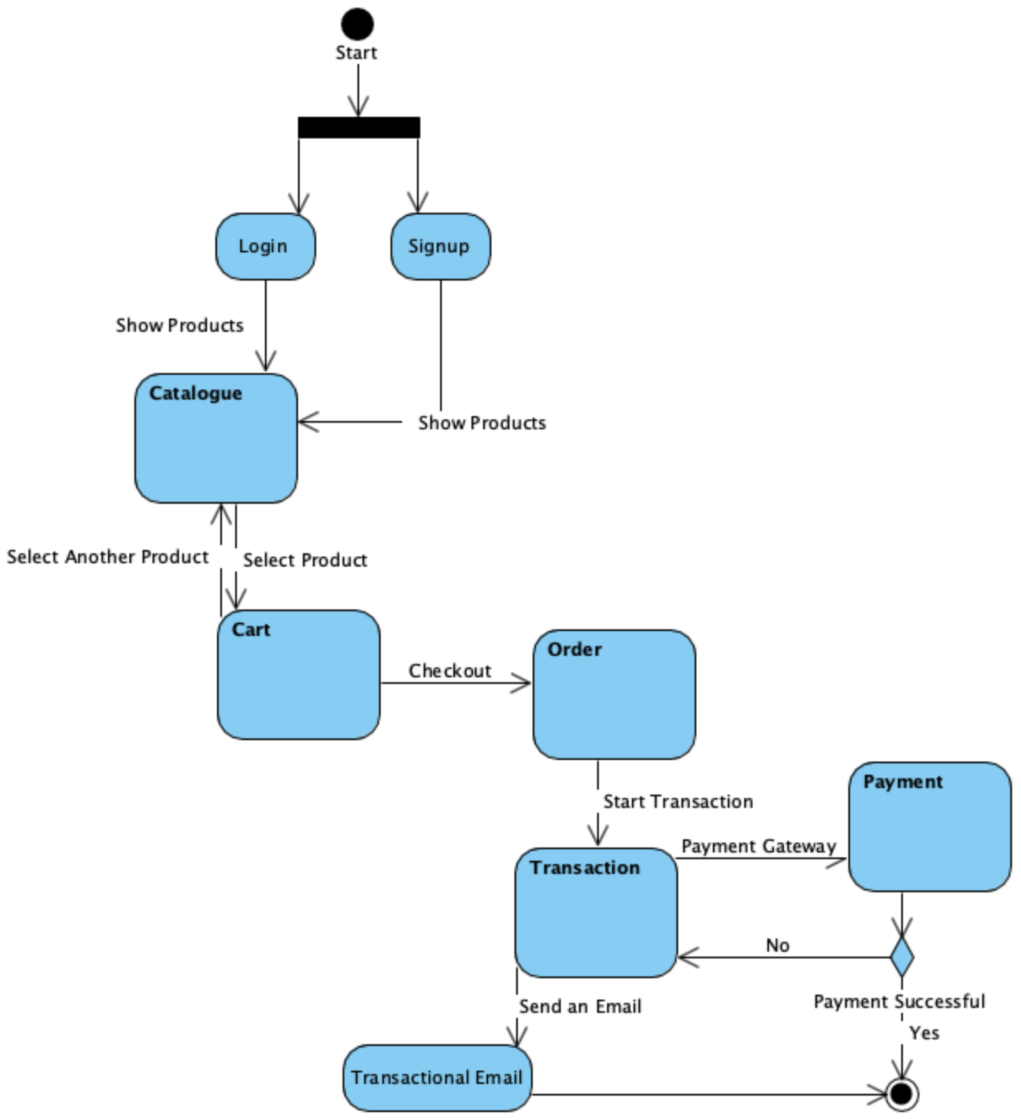
Class Diagram

A comprehensive class diagram defining the different classes present inside the package org.gdgu.cse2017.osms-api specifically for use with the REST API has been provided beneath.



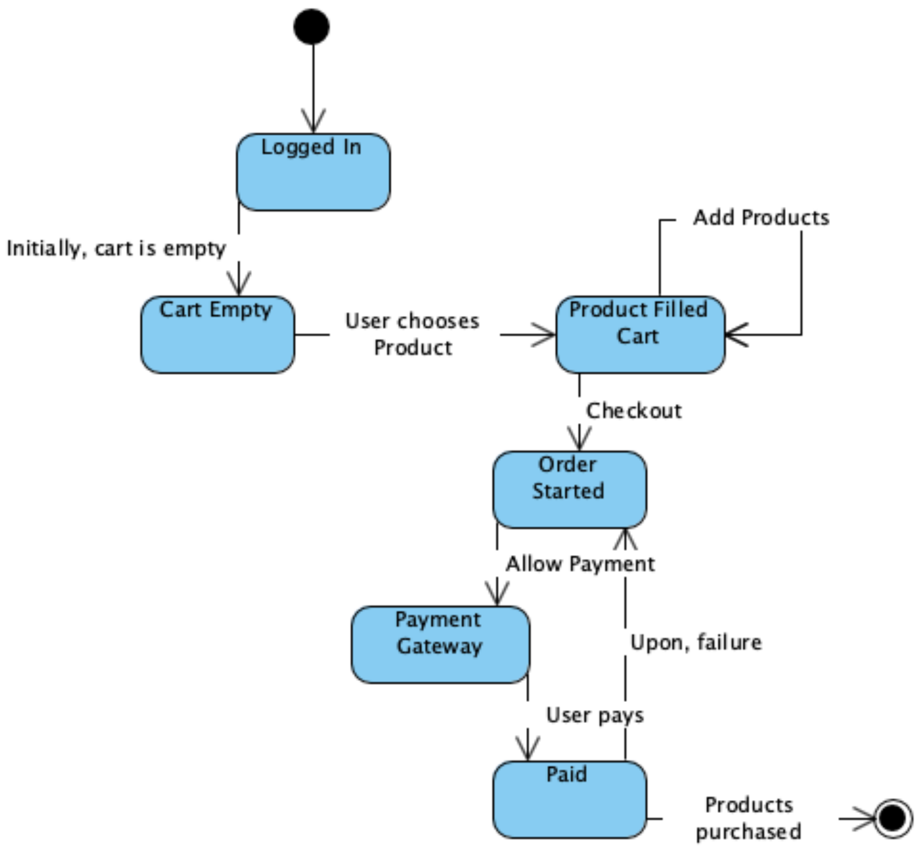
Activity Diagram

An activity diagram of one simple scenario for the application has been put beneath.



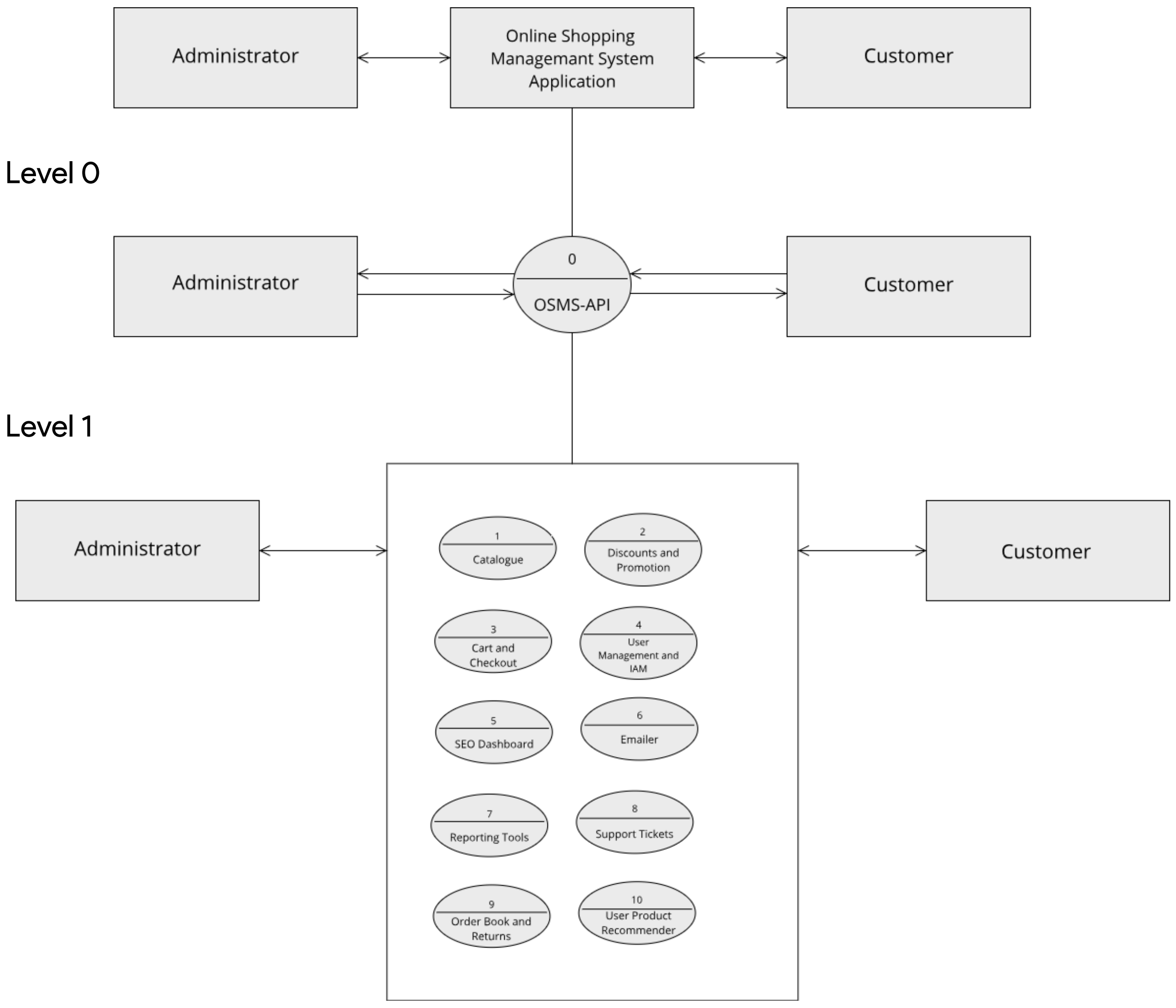
State Chart Diagram

A state chart diagram of one simple scenario for the application has been put beneath.



Data Flow Diagram

A data flow diagram for the project has been put beneath. It is a very high level view as only level 0 and level 1 diagrams have been included.



Implementation

Online Shopping Management System
v1.0

API Endpoints

url: https://cse2017-osms.herokuapp.com

- {url}/api/users/administrators/login
    - Administrator login functionality
    - Input Method: **POST**
    - Input Body: `{"username": "test.name", "password": "test_password"}`
    - Output: `true | false`
- {url}/api/users/customers/login
    - Customer login functionality
    - Input Method: **POST**
    - Input Body: `{"username": "test.name", "password": "test_password"}`
    - Output: `true | false`
- {url}/api/users/customers/signup
    - Customer signup functionality
    - Input Method: **POST**
    - Input Body: `{"name": "FirstName LastName", "email": "someone@example.com", "username": "test.name", "password": "test_password", "phone": 0000000000, "address": "Street Name, City, Country"}`
    - Output: `{**same as input**} | null`

- {url}/api/catalogue
  - Catalogue as a list of products
  - Input Method: **GET**
  - Input Body: `No body`
  - Output: `[{"id": 1, "name": "Product Name 1", "category": "Some Category", "price": 50000.0, "qtyAvailable": 50}, ...]`

Source Code

org.gdgu.cse2017.osms.User

```
package org.gdgu.cse2017.osms;

public class User {
    protected String username;
    protected String password;
    protected String email;
    public User(String username, String password, String
email) {
        this.username = username;
        this.password = password;
        this.email = email;
    }
}
```

org.gdgu.cse2017.osms.User

```
package org.gdgu.cse2017.osms;

import org.gdgu.cse2017.osms.db.Database;
```

```java
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Customer extends User {
    public String name, address;
    public long phone;
    public Customer(String username, String password, String
email, String name, long phone, String address) {
        super(username, password, email);
        this.name = name;
        this.phone = phone;
        this.address = address;
    }
    public static boolean login(String username, String
password) throws SQLException {
        PreparedStatement st =
Database.conn.prepareStatement("SELECT username, password FROM
customer WHERE username = ? AND password = ?");
        st.setString(1, username);
        st.setString(2, password);
        ResultSet rs = st.executeQuery();
        int count = 0;
        while(rs.next()) count++;
        return count == 1;
    }
    public Customer signup() throws SQLException {
        PreparedStatement st =
Database.conn.prepareStatement("INSERT INTO customer
(username, password, email, name, phone, address) VALUES
(?, ?, ?, ?, ?, ?)");
        st.setString(1, username);
        st.setString(2, password);
        st.setString(3, email);
        st.setString(4, name);
        st.setLong(5, phone);
        st.setString(6, address);
        int count = st.executeUpdate();
        return ((count == 1) ? this : null);
    }
}
```

org.gdgu.cse2017.osms.Customer

```
package org.gdgu.cse2017.osms;

import org.gdgu.cse2017.osms.db.Database;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Administrator extends User {
    public String name;
    public long phone;
    public Administrator(String username, String password,
String email, String name, long phone) {
        super(username, password, email);
        this.name = name;
        this.phone = phone;
    }
    public static boolean login(String username, String
password) throws SQLException {
        PreparedStatement st =
Database.conn.prepareStatement("SELECT username, password FROM
administrator WHERE username = ? AND password = ?");
        st.setString(1, username);
        st.setString(2, password);
        ResultSet rs = st.executeQuery();
        int count = 0;
        while(rs.next()) count++;
        return count == 1;
    }
}
```

org.gdgu.cse2017.osms.Product

```
package org.gdgu.cse2017.osms;

public class Product {
    public long id;
    public String name, category;
    public float price;
    public int qtyAvailable;
    public Product(long id, String name, float price, int
qtyAvailable, String category) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.qtyAvailable = qtyAvailable;
        this.category = category;
    }
}
```

org.gdgu.cse2017.osms.Catalogue

```
package org.gdgu.cse2017.osms;

import org.gdgu.cse2017.osms.db.Database;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;

public class Catalogue {
    public static ArrayList<Product> products = new
ArrayList<>();
    static {
        try {
            Statement st = Database.conn.createStatement();
            ResultSet rs = st.executeQuery("SELECT id, name, 
price, qty, category FROM product");
            while(rs.next()) {
                Product product = new Product(
                    rs.getLong("id"),
                    rs.getString("name"),
                    rs.getFloat("price"),
                    rs.getInt("qty"),
                    rs.getString("category")
                );
                products.add(product);
            }
        }
        catch (SQLException ex) {
            System.err.println(ex.getMessage());
        }
    }
}
```

org.gdgu.cse2017.osms.db.Database

```
package org.gdgu.cse2017.osms.db;

import java.sql.*;

public class Database {

    public static Connection conn;

    static {
        try {
            Class.forName("org.postgresql.Driver");
            conn = DriverManager.getConnection(
                    System.getenv("JDBC_DATABASE_URL"),
                    System.getenv("JDBC_DATABASE_USERNAME"),
                    System.getenv("JDBC_DATABASE_PASSWORD")
            );
        }
        catch(ClassNotFoundException ex) {
            System.err.println(ex.getMessage());
        }
        catch(SQLException ex) {
            System.err.println(ex.getMessage());
        }
    }
}
```

org.gdgu.cse2017.osms.api.AdministratorLoginServlet

```
package org.gdgu.cse2017.osms.api;

import com.google.gson.Gson;
import org.gdgu.cse2017.osms.Administrator;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;

@WebServlet("/api/users/administrators/login")
public class AdministratorLoginServlet extends HttpServlet {
    class Credentials {
        String username, password;
    }

    @Override
    public void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws IOException {
        Gson gson = new Gson();
        Credentials creds = gson.fromJson(request.getReader(),
Credentials.class);
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();
        boolean isLoggedIn = false;
        try {
            isLoggedIn = Administrator.login(creds.username, 
creds.password);
        }
        catch (SQLException ex) {
            System.err.println(ex.getMessage());
        }
        finally {
            out.println(gson.toJson(isLoggedIn));
        }
        out.close();
    }

}
```

org.gdgu.cse2017.osms.api.CustomerLoginServlet

```
package org.gdgu.cse2017.osms.api;

import com.google.gson.Gson;
import org.gdgu.cse2017.osms.Customer;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;

@WebServlet("/api/users/customers/login")
public class CustomerLoginServlet extends HttpServlet {
    class Credentials {
        String username, password;
    }

    @Override
    public void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws IOException {
        Gson gson = new Gson();
        Credentials creds = gson.fromJson(request.getReader(),
Credentials.class);
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();
        boolean isLoggedIn = false;
        try {
            isLoggedIn = Customer.login(creds.username,
creds.password);
        }
        catch (SQLException ex) {
            System.err.println(ex.getMessage());
        }
        finally {
            out.println(gson.toJson(isLoggedIn));
        }
        out.close();
    }

}
```

org.gdgu.cse2017.osms.api.CustomerSignupServlet

```
package org.gdgu.cse2017.osms.api;

import com.google.gson.Gson;
import org.gdgu.cse2017.osms.Customer;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
```

```
@WebServlet("/api/users/customers/signup")
public class CustomerSignupServlet extends HttpServlet {
    @Override
    public void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws IOException {
        Gson gson = new Gson();
        Customer provCust = gson.fromJson(request.getReader(),
Customer.class);
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();
        try {
            String jsonRepr = gson.toJson(provCust.signup());
            out.println(jsonRepr);
        }
        catch (SQLException ex) {
            System.err.println(ex.getMessage());
            out.println(gson.toJson(null));
        }
        out.close();
    }
}
```

org.gdgu.cse2017.osms.api.CatalogueServlet

```
package org.gdgu.cse2017.osms.api;

import com.google.gson.Gson;
import org.gdgu.cse2017.osms.Catalogue;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
```

```java
@WebServlet("/api/catalogue")
public class CatalogueServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws IOException {
        Gson gson = new Gson();
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();
        out.println(gson.toJson(Catalogue.products));
        out.close();
    }
}
```

Test Document

Online Shopping Management System
v1.0

Test Cases

API Endpoint 1: {url}/api/users/administrators/login

API Endpoint 2: {url}/api/users/customers/login

API Endpoint 3: {url}/api/users/customers/signup

API Endpoint 4: {url}/api/catalogue

API Testing Tool: **Postman** (https://getpostman.com)

Tests for valid **HTTP Status** for each API Endpoint with expected HTTP method

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU01-1	Endpoint 1 returns HTTP status	`POST, {}`	200 OK	200 OK	✅ Pass
TU02-1	Endpoint 2 returns HTTP status	`POST, {}`	200 OK	200 OK	✅ Pass
TU03-1	Endpoint 3 returns HTTP status	`POST, {}`	200 OK	200 OK	✅ Pass
TU04-1	Endpoint 4 returns HTTP status	`GET, None`	200 OK	200 OK	✅ Pass

Tests for valid **JSON Body** for each API Endpoint with **expected HTTP method**

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU01-2	Endpoint 1 has valid HTTP body	`POST, {}`	Any JSON	`"null"`	✅ Pass
TU02-2	Endpoint 1 has valid HTTP body	`POST, {}`	Any JSON	`"null"`	✅ Pass
TU03-2	Endpoint 1 has valid HTTP body	`POST, {}`	Any JSON	`"null"`	✅ Pass

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU04-2	Endpoint 1 has valid HTTP body	`GET, None`	Any JSON	`"[{product}]"`	✅ Pass

<u>Tests for valid **JSON Body** for each API Endpoint with **unexpected HTTP methods**</u>

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU01-3	Endpoint 1 produces valid body with GET	`GET, None`	Any JSON	`"<!DOCTY.."`	❌ Fail
TU02-3	Endpoint 2 produces valid body with GET method	`GET, None`	Any JSON	`"<!DOCTY.."`	❌ Fail
TU03-3	Endpoint 3 produces valid body with GET method	`GET, None`	Any JSON	`"<!DOCTY.."`	❌ Fail
TU04-3	Endpoint 3 produces valid body with POST method	`POST, {}`	Any JSON	`"<!DOCTY.."`	❌ Fail

<u>Tests for functionality **Administrator Login** at API endpoint 1</u>

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU01-4	Valid username and password	`POST, {"username": "devka", "password": "iamadmin123"}`	`"true"`	`"true"`	✅ Pass
TU01-5	Invalid username and password	`POST, {"username": "aryana", "password": "manpower"}`	`"false"`	`"false"`	✅ Pass

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU01-6	Valid username and invalid password	`POST, {"username": "devka", "password": "kirl"}`	`"false"`	`"false"`	✅ Pass
TU01-7	Invalid data type for username	`POST, {"username": 123, "password": "kirl"}`	`"null"`	`"false"`	❌ Fail
TU01-8	Invalid data type for password	`POST, {"username": "devka", "password": 123}`	`"null"`	`"false"`	❌ Fail
TU01-9	Invalid input	`{"name": 1}`	`"null"`	`"false"`	❌ Fail

Tests for functionality **Customer Login** at API endpoint 2

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU02-4	Valid username and password	`POST, {"username": "paditi", "password": "smile456"}`	`"true"`	`"true"`	✅ Pass
TU02-5	Invalid username and password	`POST, {"username": "aryana", "password": "manpower"}`	`"false"`	`"false"`	✅ Pass
TU02-6	Valid username and invalid password	`POST, {"username": "paditi", "password": "kirl"}`	`"false"`	`"false"`	✅ Pass
TU02-7	Invalid data type for username	`POST, {"username": 123, "password": "kirl"}`	`"null"`	`"false"`	❌ Fail

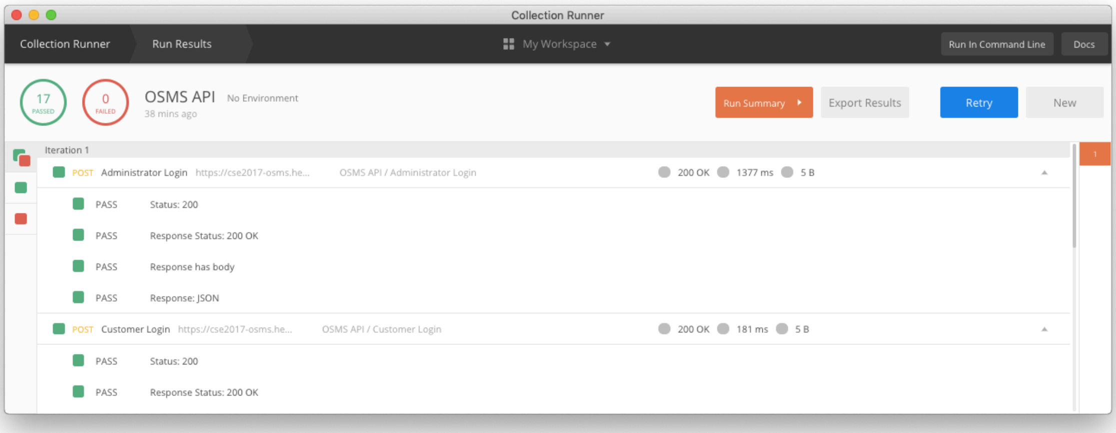
Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU02-8	Invalid data type for password	`POST, {"username": "devka", "password": 123}`	`"null"`	`"false"`	❌ Fail
TU02-9	Invalid input	`{"name": 1}`	`"null"`	`"false"`	❌ Fail

<u>Tests for functionality **Customer Signup** at API endpoint 3</u>

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU03-4	Signup with valid user details	`POST, {"name": "Akshay Tripathy", "email": "ak6y@e.com", "username": "aktrip", "password": "passwd", "phone": 8961512762, "address": "Gurgaon, India"}`	Same as input	`{"name": "Akshay Tripathy", "email": "ak6y@e.com", "username": "aktrip", "password": "passwd", "phone": 8961512762, "address": "Gurgaon, India"}`	✅ Pass
TU03-5	Signup with incomplete user details	`POST, {"name": "Anush Verma", "email": "triagon"}`	`"null"`	`"null"`	✅ Pass
TU03-6	Signup for username that already exists	`POST, {"name": "Akshay Tripathy", "email": "ak6y@e.com", "username": "aktrip", "password": "passwd", "phone": 8961512762, "address": "Gurgaon, India"}`	`"null"`	`"null"`	✅ Pass

Tests for functionality **Catalogue** at API endpoint 4

Test Case ID	Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail
TU04-4	List of products from catalogue	`GET, None`	`[{product}]`	`[{"id": 1, "name": "Apple MacBook Air", "category": "Electronics", "price": 50000.0, "qtyAvailable" :50}, {"id": 2, "name": "OnePlus 6T", "category": "Electronics", "price": 30000.0, "qtyAvailable": 20},...]`	✅ Pass
TU04-5	Each product in catalogue follows a schema	`GET, None`	Follows schema: `[{"id": long, "name": "string", "category": "string", "price": float, "qtyAvailable": int}, ...]`	`Same as above`	✅ Pass



17 PASSED | 0 FAILED

OSMS API No Environment
38 mins ago

Iteration 1

```
POST  Administrator Login  https://cse2017-osms.he...   OSMS API / Administrator Login   200 OK  1377 ms  5 B
      PASS   Status: 200
      PASS   Response Status: 200 OK
      PASS   Response has body
      PASS   Response: JSON
POST  Customer Login  https://cse2017-osms.he...   OSMS API / Customer Login   200 OK  181 ms  5 B
      PASS   Status: 200
      PASS   Response Status: 200 OK
```

All results tabulated above were performed using Postman (an API testing tool) collections runner that uses Chai-based scripting for writing test cases. The results collected are from Postman collections runner.

Bug Tracker

GitHub Issues

Bug Tracker Tool

Bug Trackers are software programs that are generally integrated with other development tools and used as a part of the software development life cycle to keep a track of bugs that are being faced by the users with respect to a particular software. In case of major software projects these bugs are reported by the people who engineer the software and in some cases it may be directly the people using that particular software. Most bug trackers also feature a functionality that allows the development team to mark which bug have been fixed and provide different information regading the progress of the bug fix.

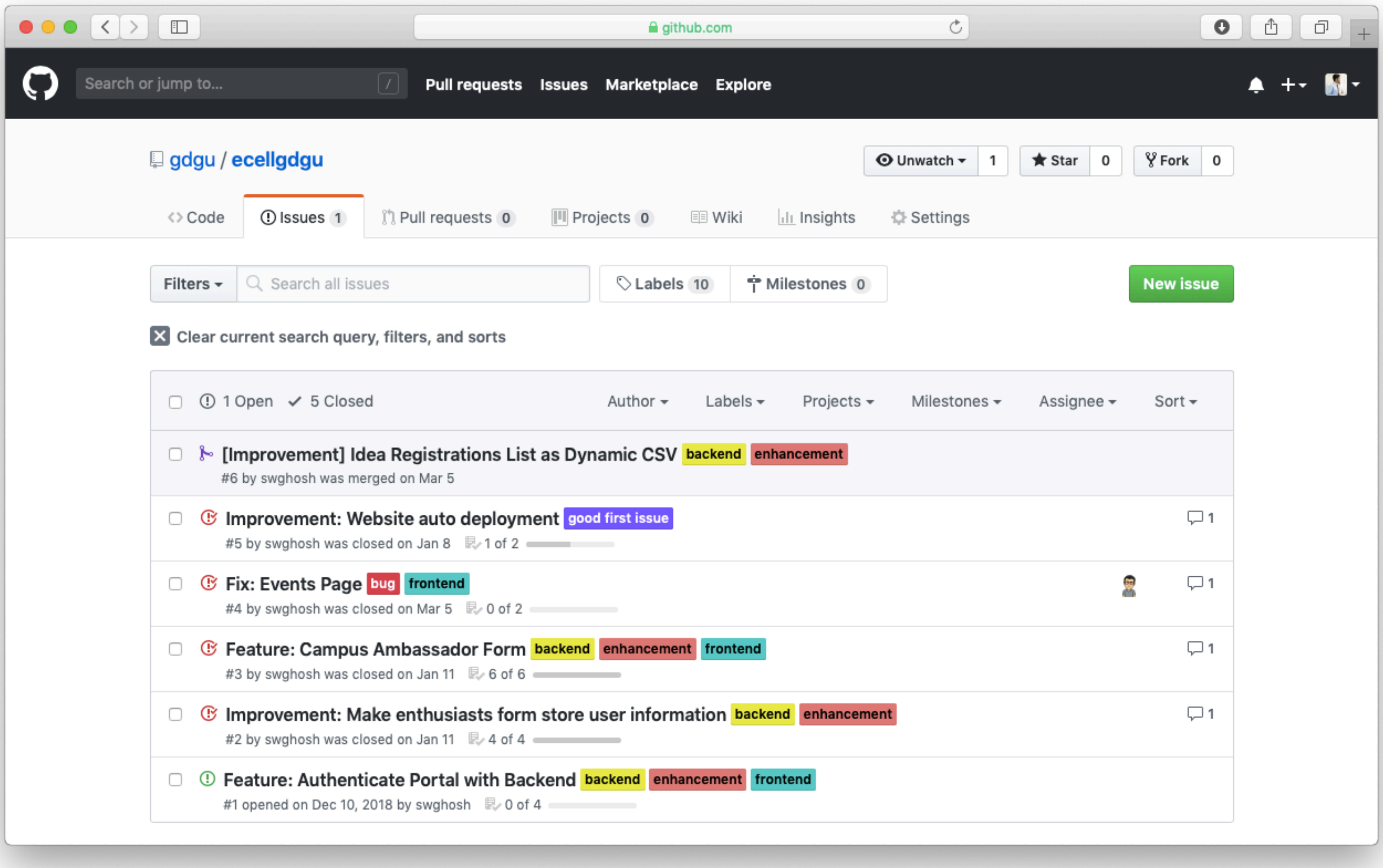
During enterprise oriented software development it is notable that these bugs are reported by the members of the organisation involved in engineering the software on the basis of either user reviews or bugs faced during the testing of the software. Bug trackers are also known to be useful as an incident management system to keep a chronological records of different bugs and failures faced by the software product.

While in case of open source software development a majority of bugs are reported by the user itself. The community behind the software are responsible for fixing the different bugs reported which may be assigned on an individual basis by the project maintainers.

GitHub Issues

GitHub Issues provides an elegant interface for tracking bugs across software projects. It is a part of the GitHub ecosystem that hosts a million open source projects. Every software on GitHub is hosted in the form of a repository which also keeps track of the source code for the software project with the help of Git version control system. Any repository on GitHub provides the Issues functionality which is in the form of a bug tracker. Each repository may either be public or private.

Public repositories are used for open source software projects while private repositories are used for maintaining closed source, proprietary/enterprise-driven software. The Issues functionality is complemented with the Pull Requests functionality which is used to submit fixes for a particular bug that has been reported on the system.


github.com

Search or jump to... Pull requests Issues Marketplace Explore

gdgu / ecellgdgu

Unwatch 1 Star 0 Fork 0

Code Issues 1 Pull requests 0 Projects 0 Wiki Insights Settings

Filters Search all issues Labels 10 Milestones 0 New issue

Clear current search query, filters, and sorts

1 Open ✓ 5 Closed Author Labels Projects Milestones Assignee Sort

- [Improvement] Idea Registrations List as Dynamic CSV `backend` `enhancement`
  #6 by swghosh was merged on Mar 5
- Improvement: Website auto deployment `good first issue` — 1
  #5 by swghosh was closed on Jan 8 — 1 of 2
- Fix: Events Page `bug` `frontend` — 1
  #4 by swghosh was closed on Mar 5 — 0 of 2
- Feature: Campus Ambassador Form `backend` `enhancement` `frontend` — 1
  #3 by swghosh was closed on Jan 11 — 6 of 6
- Improvement: Make enthusiasts form store user information `backend` `enhancement` — 1
  #2 by swghosh was closed on Jan 11 — 4 of 4
- Feature: Authenticate Portal with Backend `backend` `enhancement` `frontend`
  #1 opened on Dec 10, 2018 by swghosh — 0 of 4

An example of a GitHub repository's Issues tab have been put above that shows some of the bugs reported and fixed for that software project.