Dowling, Inc.
Bike Rental Web Application

Team H

Reed Ladnier (Team Leader)
Sneha Gore, Alexander Ford, Jared Key, Jarius Williams, Timothy Lee

Spring 2021

Table of Contents

Preface

The following document is designed to demonstrate the requirements and specifications for the web system requested by Dowling, Inc to manage and facilitate the renting of bikes by customers of Dowling, Inc. This will serve as the specification agreement between Group H and Dowling, Inc. for the purpose of building the web application and should be agreed upon by both parties.

This is version 3 of the document and serves as the system design for the bike rental web application prototype.

Introduction

The bike rental application provides a wide range of features that allow customers and staff to efficiently interact with Dowling inc. services. This system is being developed to overcome manual errors that occur during reservation and imposing fees based on the time consumed by the user. The application also provides advanced reports which help the owner analyze data regarding bike rental usage. Any detailed information regarding availability of bikes at docks, charges, consumption time can be obtained easily with high accuracy in no time. It also eliminates the need of labour, infrastructure and time taken by manual work in the traditional system.

The system is user friendly which provides flexibility for users to book bikes ensuring end to end payment security along with data integrity . Booking from the application automates the manual process of booking bikes based on the availability at the dock station, calculation of tariff as per user's time period, and cashless secured payment. The user details are stored in application forms with respective payment mode information while maintaining privacy of the information. This is very beneficial to the user for further reference and it prevents the user from doing redundant reservation steps. There are various different modules of this application that can be accessed and managed by the owner and manager. The application also generates reports which helps to measure business performance and draw business insights based on the reported numbers. This robust system can be modified to accommodate new bikes, and dock stations as per increasing business needs.

Glossary

Validation: The process of the browser and/or the web server will check to see that the data is in the correct format and within the constraints set by the application.
Client-Server architecture: A computer network where clients receive service from a centralized server.
React JS: React JS is a javascript library used to create user interfaces for web applications.
Express:

User Requirements Specification

Stories or Scenarios

1. As Mr. Dowling, I want to edit rates for bikes so that I can control the prices for rental
   Given I log onto the application as the owner
   I am allowed to change the prices for all docks
2. As Mr. Dowling, manager, or customer can access from mobile and desktop so that I can access from anywhere on mobile or desktop
   Given I go to the application on phone or desktop
   Then it is formatted correctly
3. As a customer, I would like to see dock locations
   Given I log onto the application
   Then dock locations are shown on a map
4. As a manager, I want automatic charging when returned
   Given the given bike is returned
   Then the card is charged
5. As a manager, I want credit card validation before bike use
   Given credit card information is entered
   Then validation status is shown
6. As Mr. Dowling, I want customers to have an account for continued use
   Given a customer wants a bike rental
   An account must be created
7. As a manager, I want to track how many bikes are at each dock
   Given a manager signs in to their account
   Then the number of bikes at each dock is displayed
8. As Mr. Dowlilng wants a weekly report for income and bike usage
   Given Mr. Dowling logs in
   Then weekly report for income and bike usage is shown

Use cases

1. Actor Mr. Dowling Use Case: Weekly Report
   Rationale: Mr. Dowling wants to see his weekly report
   Actor: Mr. Dowling
   Preconditions: Week has passed
   Standard Path:
   1. Mr. Dowling logs into the application.

2. The weekly report is shown

Alternate Path: N/A

Postconditions: N/A

2. Actor Mr. Dowling Use Case: Price modifications

    Rationale: Mr. Dowling wants to be able to change prices/fees/rates for bike rentals

    Actor: Mr. Dowling

    Preconditions: N/A

    Standard Path:

    1. Mr. Dowling logs into the application
    2. Pulls up prices
    3. Modifies prices

    Alternate Path: N/A

    Postconditions: Prices are modified

3. Actor Manager Use Case: Card validation for bike unlocks

    Rationale: The manager wants to be able to see validation status before unlocking the bike for rental

    Actor: Manager

    Preconditions: Credit card information is given

    Standard Path:

    1. Manager logs into the application
    2. Manager gathers customer information
    3. Manager checks valid credit card is on file
    4. Manager unlocks bike for a customer

    Alternate Path: In step 3, the customer does not have a valid credit card. Informs customer.

    Postconditions: N/A

4. Actor Manager Use Case: Record damages to bike for repair fees

    Rationale: The manager wants to be able to enter the level of damage on the bike to accurately charge the customer for damage

    Actor: Manager

    Preconditions: The returned bike is seen as damaged

    Standard Path:

    1. Manager logs into the application
    2. Manager checks bikes returned for damage
    3. The manager adds repair fees onto the customer's bill

    Alternate Path: N/A

    Postconditions: Customer charged the repair fee and sent and informed via email.

5. Manager/Customer Use Case: Check number of bikes in dock

Rationale: Manager and customer would be able to check how many bikes are available for renting
Actor: Manager/Customer
Preconditions: N/A
Standard Path:
1. The Manager/Customer logs into the application
2. Manager/Customer checks dock to see number of bikes available

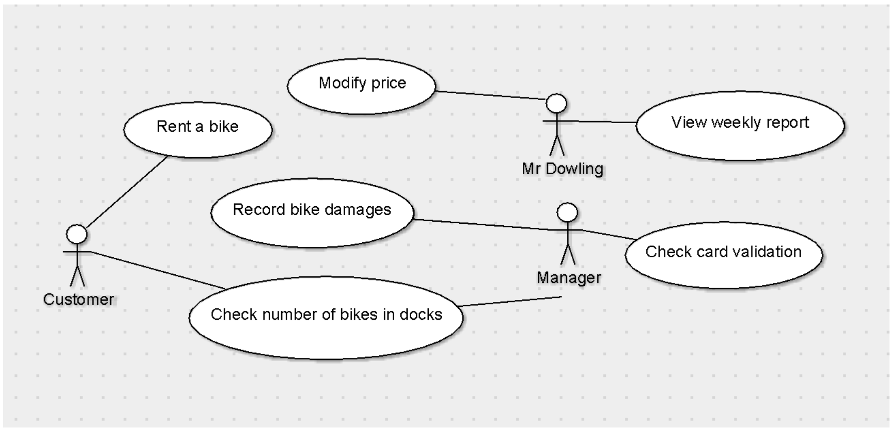
Postconditions: N/A

- Customer Use Case: Renting a bike

Rationale: Customer wants to rent a bike for quick travel around town
Actor: Customer
Preconditions: Bikes are loaded in the docks and ready for renting
Standard Path:
1. The customer opens the website and logs in.
2. Customer requests a rental from the dock they are at
3. The customer takes the bike.
4. The customer returns the bike to a dock.
5. The customer is charged the fee.

Alternate Path 1: In step 1, if the customer has no account, they must create an account before the rental request is submitted and continues on to step 2.
Alternate Path 2: In step 4, if the bike is not returned successfully, a customer is charged an alternate fee.
Postconditions: Bike counts of each dock are updated to reflect the new number of bikes.

Use Cases Diagram



Project Backlog

1. Generate Weekly Report
    - 1.1 Show total bike usage
    - 1.2 Calculate income of each dock
2. Account Creation/Login Web pages
    - 2.1 Store user credentials
        - 2.1.1 Address
        - 2.1.2 Birthdate
        - 2.1.3 Phone number
        - 2.1.4 Credit card information
        - 2.1.5 Email address
        - 2.1.6 Validate credit card and email address
        - 2.1.7 Validate user login
3. Bike Rental Rates Calculator
    - 3.1 Calculates rental fee when returned
        - 3.1.1 $5 per half hour
        - 3.1.2 $9 per hour, portion of hour used charged full price of hour
        - 3.1.3 $25 if returned to different dock
        - 3.1.4 $500 if bike is not returned
        - 3.1.5 $200 if bike is damaged depending on damage
4. Docks/Bikes Webpage
    - 4.1 Uniquely identify each of the 10 docks
        - 4.1.1 Track number of bikes at each dock, not exceeding 6 bikes per dock
5. Customer Agreement Webpage
    - 5.1 Must indicate knowledge and agreement to requirements
        - 5.1.1 Rates
        - 5.1.2 Account creation
        - 5.1.3 Penalties
        - 5.1.4 Features

System Architecture

In order to facilitate the web access to the bike rental system by customers and ensure safe storage of users information, a client-server architecture has been decided. The client-server architecture will allow easy maintenance and security of centralized systems for Dowling, Inc. following system completion.

The client will be built using ReactJS to facilitate fast rendering on a multitude of devices and strong community support for easier continued management. Typescript will be used to enforce proper typing and encourage clean code to be written. Requests to the server will be made using the package Axios to create HTTP requests. Styling for the client will use the UI library Material-UI for fast and consistent prototyping.

The server will be built using ExpressJS for the customizability and convenience of a single language for both client and server. Typescript will also be used for the server to enforce proper typing and clean code. The database used will be MySQL connecting to the ExpressJS app through a promisified mysql node module. Bcrypt will be used to provide hashes and salting for passwords stored in the database to ensure security of user information.

System Requirements Specification

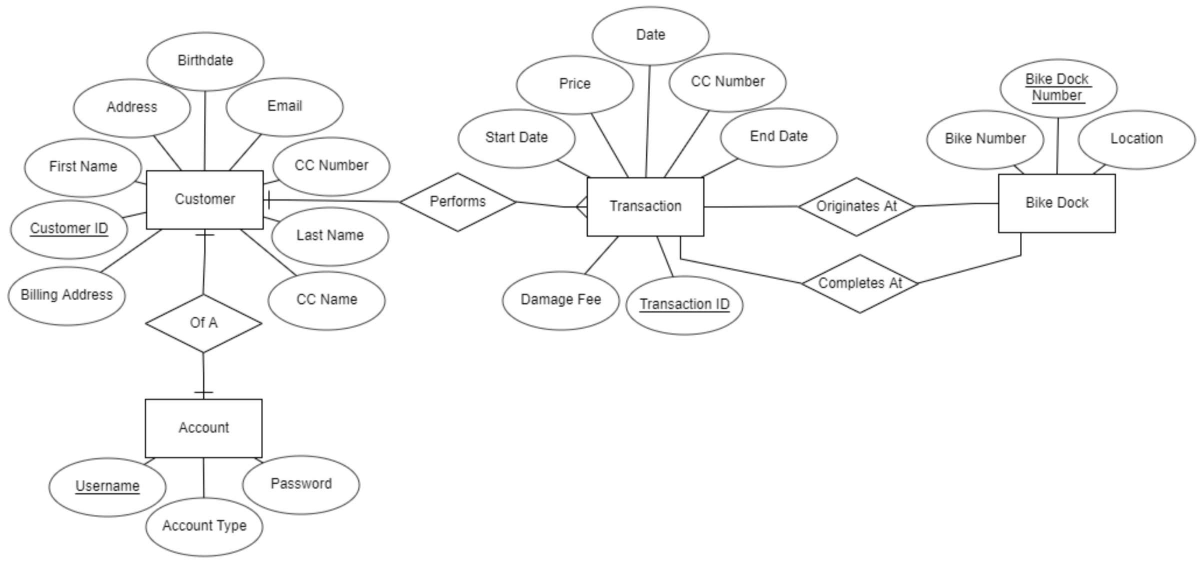
1. Account creation for users
    1.1. Webpages for signup
    1.2. Acknowledgment of the terms & conditions
    1.3. Manual creation of manager accounts from owner account
    1.4. Database for user/manager accounts
        1.4.1. Account addition to the database
        1.4.2. Account retrieval (credit card information and name)
        1.4.3. Ability to edit address/phone number/credit card
            1.4.3.1. Web forms inside user account view to request edits
            1.4.3.2. Validate information for users before updating
            1.4.3.3. Allow storing of multiple credit cards
        1.4.4. Updating/Deleting User login
    1.5. Webpages for user login
    1.6. Email validation
    1.7. Password validation
    1.8. Forgot password option
2. Credit card validation
    2.1. Correct format check
3. Bike rental price editor (Owner)
4. Webpage with Owner authorization
5. Update charge module with new information and update user view to display new rates.
    5.1. (Retain old rate for users currently renting)
    5.2. Updates map dock stations with new prices
6. Auto-updating weekly report system (Owner)
7. Webpage with admin authorization
    7.1. Collect all dock and rental information from the previous week on a Sunday
    7.2. Analyze/format data into rental rates and income of each dock
    7.3. Display a notification on the owner account to indicate that the weekly report is ready
8. Fee calculation module
    8.1. Rate card (hourly)
    8.2. Charges for damage caused
    8.3. Charges for non-return of bike
    8.4. Manager view page for updating fee
9. Bike counter at each dock
    9.1. Max bikes (6) at dock notification
    9.2. Bikes available notification
10. Warn of a fee increase for different dock return

- 10.1. Record which docks the customer rented from

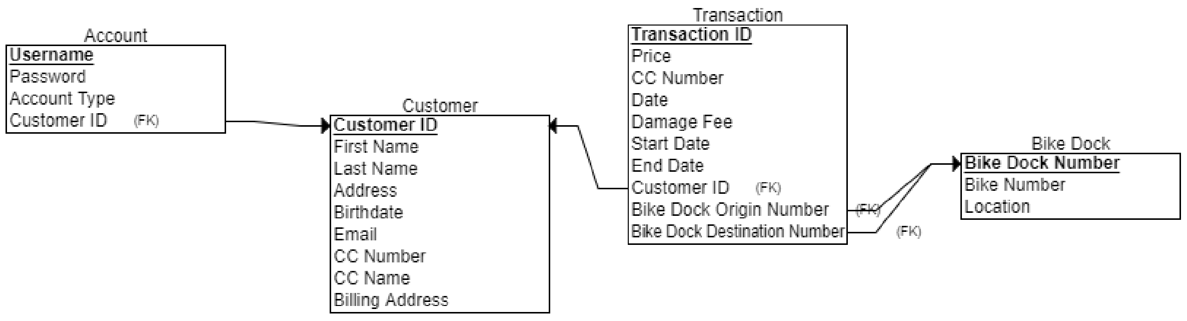
11. Mobile-safe formatting
12. Verify all web pages format correctly on both mobile and desktop version of the site
13. Different views for different account levels
14. User view
    - 14.1. Show dock locations/number of bikes and rental fees
    - 14.2. Allow user to rent bike
    - 14.3. Allowed edit access to user accounts
15. Owner view
    - 15.1. Contains easy access to the weekly report and monitoring information for bike docks (usage + income)
    - 15.2. Allowed edit access to manager accounts
    - 15.3. Price editor
16. Manager view
    - 16.1. Contains information of customers as they arrive to dock and request rental and as they return bikes
    - 16.2. Availability of the bikes
17. Backend Server
    - 17.1. Methods for user account creation and account login
    - 17.2. Methods for manager account creation and login
    - 17.3. Methods for updating bike counters
18. Database design, and creation

System Models

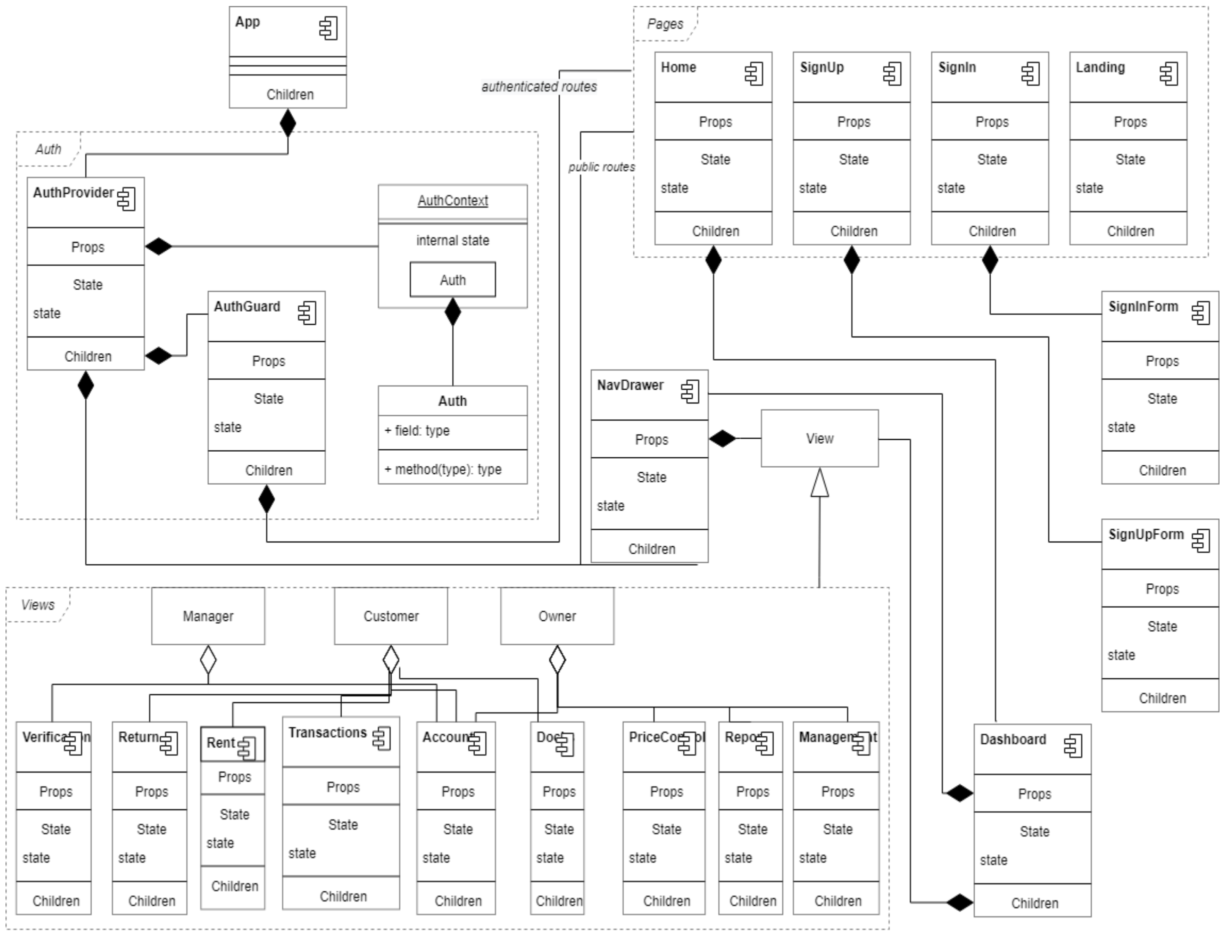
Database ER Diagram:



Database Relational Model:



React Component Model:



System Evolution

Not Applicable.

Appendices

Not Applicable.

Index

Dowling, Inc.
Bike Rental Web Application
Team H

Test Plan
Version 0.2

Revision History

Version	**Date**	**Author**	**Description**
V0.1	03/10/2021	Group H Team members	Initial Draft
V0.2	03/24/2021	Group H Team members	Added Unit Test Results under 8.1 Test Results section.

Document References

Document	**Version**	**Date Released**
Requirement Document	0.2	02/23/2021

Table of Contents

1. Introduction

The Dowling Inc Bike Rental Application will provide a wide range of features that allow customers and staff to efficiently interact with Dowling inc. services.It provides advanced reports, availability information.The testing of this application includes end to end testing along with documenting test artefacts like Test Plan , Test Results.

1.1. Test Plan Objectives

The Test Plan has been created to document the testing activities in this project. It includes the objectives,test strategy, schedule, deliverables and approach required to perform testing for Dowling, Inc. Bike Rental Web Application.This document will clearly identify what is deemed in and out of scope with respect to testing.

2. Scope

Testing scope includes all requirements specified in the requirement document.

2.1 In Scope

All the functional and non-functional requirements.

2.1.1 User Requirements:

UC001: Weekly Report
UC002: Price modifications
UC003: Card validation for bike unlocks
UC004: Record damages to bike for repair fees
UC005: Check number of bikes in dock

2.2 Out Of Scope

Modules/scenarios in application that are not mentioned in ‘In Scope’

3. Test Strategy

The test strategy consists of a series of different tests that are planned for testing the Dowling Inc bike rental application. The primary purpose of these tests is to uncover the systems limitations and measure its full capabilities. A list of the various planned tests and a brief explanation follows below.

3.1 Unit Test

The Unit tests in the project will focus on the behavior of individual components of the Dowling, Inc. bike rental web application. Unit Test will verify the functionality in the individual units of source code, computer program modules together with associated data.

3.2 Performance Test

Performance Test will emphasize on the behaviour of the system under the specified load. It includes the endurance testing of the product to verify the system parameters with expected load. It will evaluate the speed,responsiveness and stability of the product.

3.3 System Integration Test

System Integration Testing involves testing of complete system System Integration Test will test the integrated system and validate the results as per the expected results.This test will validate the interactions between the modules of the application on the integrated system.

3.4 User Acceptance Test

User Acceptance Testing will be performed once he application is ready for implementation.This testing is performed to check the developed application is according to specified user requirements and can be used by the end user.

4. Test Approach

The project is using an agile approach, with weekly iterations. At the end of each week the requirements identified for that iteration will be delivered to the team and will be tested.

Test Case Format :
Test case description:
Input:
Condition or function under test:
Expected Output:
Output:

5. Test Environment

Server is required for the web application setup and the database.

6. Project Milestones & Deliverables

6.1 Test Schedule

Testing	**Start Date**	**End Date**
Unit Testing	03/17/2021	To be Decided
Performance Testing	Not Applicable	Not Applicable
System Integration Testing	To be Decided	To be Decided
User Acceptance Testing	To be Decided	To be Decided
Regression Testing	Not Applicable	Not Applicable

6.2 Deliverables

Deliverable	**Responsible Person**	**Completion Date**
Test Plan	Testing Team	03/15/2021
Test Cases	Testing Team	03/18/2021
Defect Report	Testing Team	04/16/2021
Test Completion Report	Testing Team	04/16/2021

Testing Team : Responsible for performing the actual system integration testing and providing test artefacts.

7. Test Artefacts

8.1 Test Results

8.1.1 Unit Test Results

TC01:

Description: Create new user account

```
Input:          agent
Route:          .post('/api/accounts/signup')
                .send({
                        first_name: 'John',
                        last_name: 'Doe',
                        email: 'user3@email.com',
                        password: 'password',
                        birthdate: new Date().toISOString(),
                        cc_number: '1234567891011121',
                        billing_address:'123 N. Madeup St.',
                        address:'123 N. Madeup St.',
                        cc_name: 'John Doe',
                        username: 'dev3',
Expected:       }).then((res) => {
                        expect(res.statusCode).toEqual(200);
                        agent.jar.setCookie(res.headers['set-cookie'][0]);
                });
```

Result: Jest Test Passed

TC02:

Description: Get Signed-in user information

```
Input:          agent
Route:          .get('/api/customers/me')
Expected:       .then((res) => {
                        expect(res.statusCode).toEqual(200);
                        expect(res.body).toHaveProperty('first_name');
                        expect(res.body).toHaveProperty('last_name');
                        expect(res.body).toHaveProperty('address');
                        expect(res.body).toHaveProperty('email');
                        expect(res.body).toHaveProperty('cc_number');
                        expect(res.body).toHaveProperty('cc_name');
                        expect(res.body).toHaveProperty('billing_address');
                });
```

Result: Jest Test Passed

TC03:
Description: Sign out of Account
```
Input:      agent
Route:      .post('/api/accounts/logout')
Expected:   .then((res) => {
                expect(res.statusCode).toEqual(200);
            });
```
Result: Jest Test Passed

Description: Login to account
```
Input:      agent
Route:      .post('/api/accounts/login')
            .send({
                 password: 'password',
                username: 'user1',
            })
Expected:   .then((res) => {
                expect(res.statusCode).toEqual(200);
                agent.jar.setCookie(res.headers['set-cookie'][0]);
            });
```
Result: Jest Test Passed

TC04:
Description: Rent a bike
```
Input:      agent
Route:      .post('/api/transactions/rent')
            .send({
                dock: 1
            })
Expected:   .then((res) => {
                expect(res.statusCode).toEqual(200);
                expect(res.body).toHaveProperty('transaction_id');
            })
```
Result: Jest Test Passed

TC05:
Description: Rent a bike from a dock with no bikes
Input: agent
Route:
```
.post('/api/transactions/rent')
.send({
        dock: 1
})
```
Expected:
```
.then((res) => {
        expect(res.statusCode).toEqual(400);
})
```
Result: Jest Test Passed

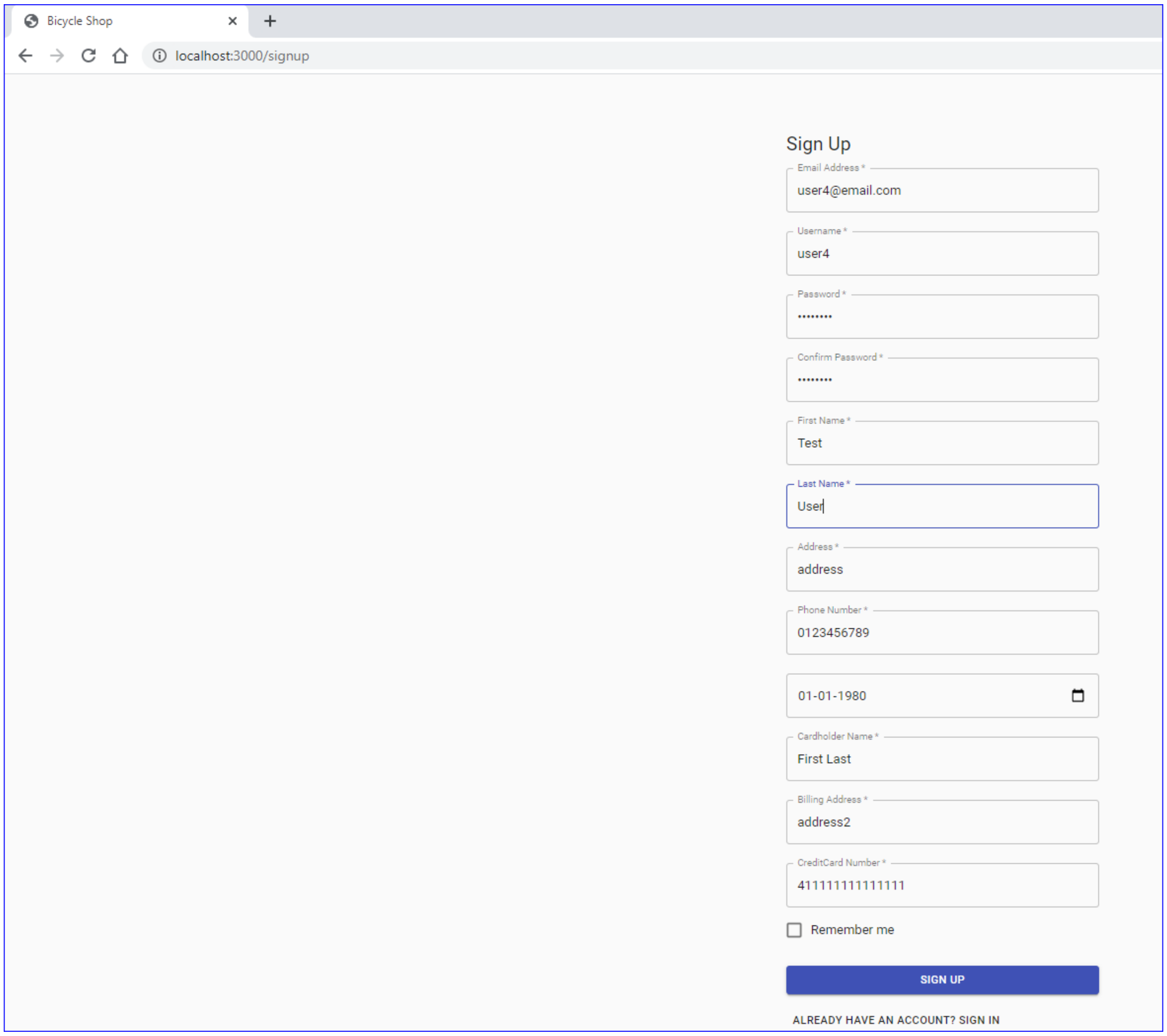
TC06:
Description: Get current transactions
Input: agent
Route:
```
.get('/api/transactions/active_transactions')
.send({})
```
Expected:
```
.then((res) => {
        expect(res.statusCode).toEqual(200);
        expect(res.body).toHaveProperty('transaction_ids');
        let list = res.body.transaction_ids;
        expect(list.length == 4);
})
```
Result: Jest Test Passed

TC07:
Description: Return a bike
Input: agent
Route:
```
.post('/api/transactions/return')
.send({
        destination_dock: 3,
        transaction_id:1
})
```
Expected:
```
.then((res) => {
        expect(res.statusCode).toEqual(200);
        expect(res.body).toHaveProperty('price');
})
```
Result: Jest Test Passed

TC08:
Description: Return a bike to a full dock
Input: agent
Route:
```
.post('/api/transactions/return')
.send({
        destination_dock: 3,
        transaction_id:4
})
```
Expected:
```
.then((res) => {
        expect(res.statusCode).toEqual(400);
})
```
Result: Jest Test Passed

8.1.2 SIT Test Results:

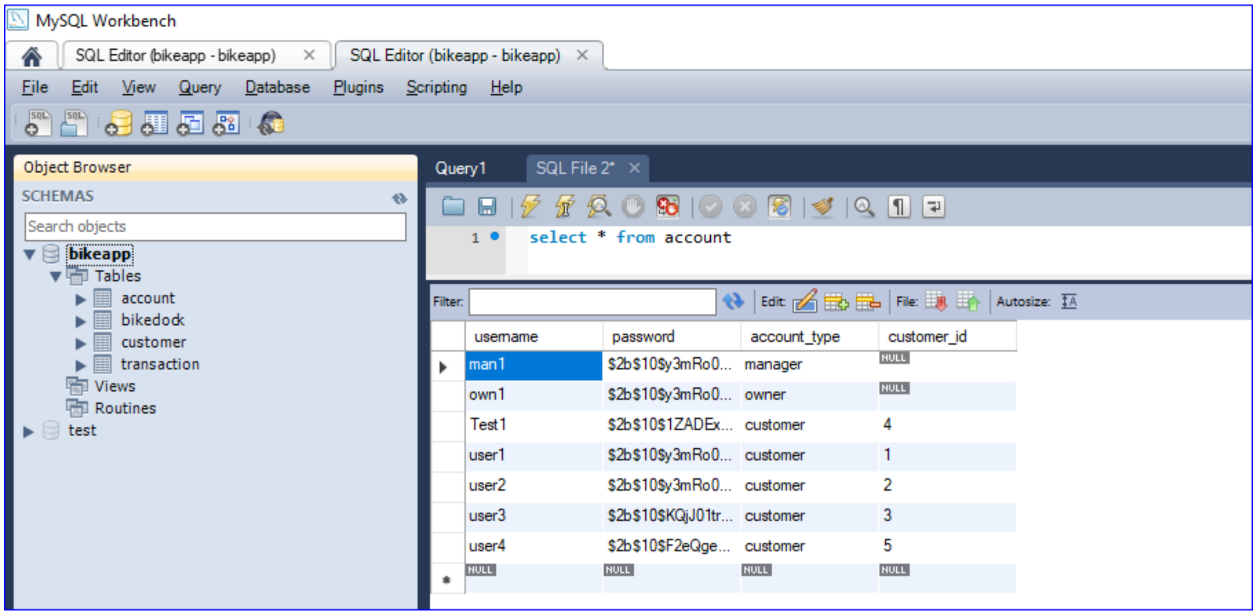
SIT_ TC01:
Test case description: To verify Signup functionality
Prerequisite : Bike App website should be correctly configured and running.
Input: User details for signing up
Condition or function under test: .post('/api/accounts/signup')
Expected Output: User should be successfully signed up
Actual Output: User is successfully signed up to the Bike App website.Newly created user is displayed in database.
Test Data: User ID : user4 , Password : password
Test Result:



Sign Up

Email Address *	user4@email.com
Username *	user4
Password *	••••••••
Confirm Password *	••••••••
First Name *	Test
Last Name *	User
Address *	address
Phone Number *	0123456789
	01-01-1980
Cardholder Name *	First Last
Billing Address *	address2
CreditCard Number *	4111111111111111

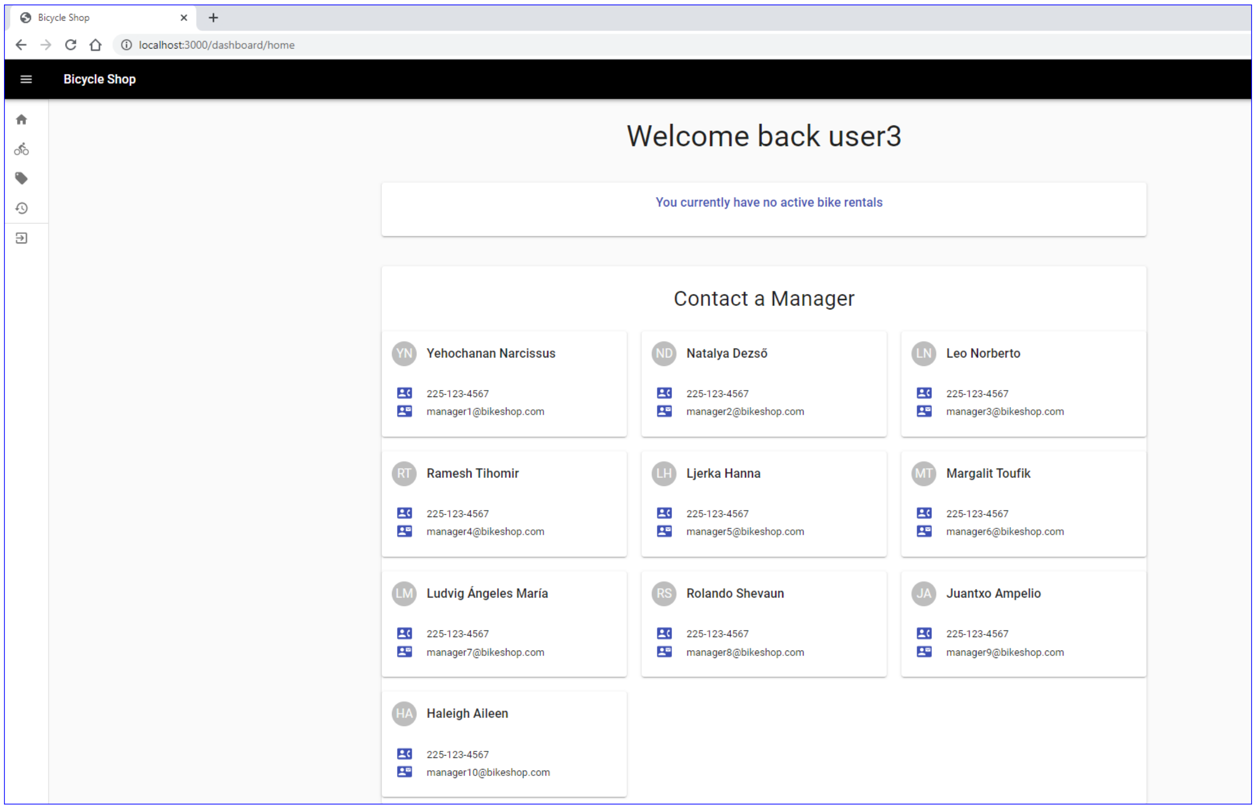
☐ Remember me



```
1 select * from account
```

	username	password	account_type	customer_id
▶	man1	$2b$10$y3mRo0...	manager	NULL
	own1	$2b$10$y3mRo0...	owner	NULL
	Test1	$2b$10$1ZADEx...	customer	4
	user1	$2b$10$y3mRo0...	customer	1
	user2	$2b$10$y3mRo0...	customer	2
	user3	$2b$10$KQjJ01tr...	customer	3
	user4	$2b$10$F2eQge...	customer	5
*	NULL	NULL	NULL	NULL

SIT_ TC02:
Test case description: To verify login functionality for User,Manager and Owner
Prerequisite : Bike App website should be correctly configured and running.
Input: User details for signing up
Condition or function under test: .post('/api/accounts/signup')
Expected Output: User should be successfully logged in to the BIke App Website.
Actual Output: User is successfully signed up to the Bike App website.
Test Data: User ID : user3 , Password : password , User ID : own1, Password : password , User ID : man1, Password : password
Test Result: for USer Login:


Bicycle Shop

localhost:3000/dashboard/home

Bicycle Shop

Welcome back user3

You currently have no active bike rentals

Contact a Manager

Yehochanan Narcissus
225-123-4567
manager1@bikeshop.com

Natalya Dezső
225-123-4567
manager2@bikeshop.com

Leo Norberto
225-123-4567
manager3@bikeshop.com

Ramesh Tihomir
225-123-4567
manager4@bikeshop.com

Ljerka Hanna
225-123-4567
manager5@bikeshop.com

Margalit Toufik
225-123-4567
manager6@bikeshop.com

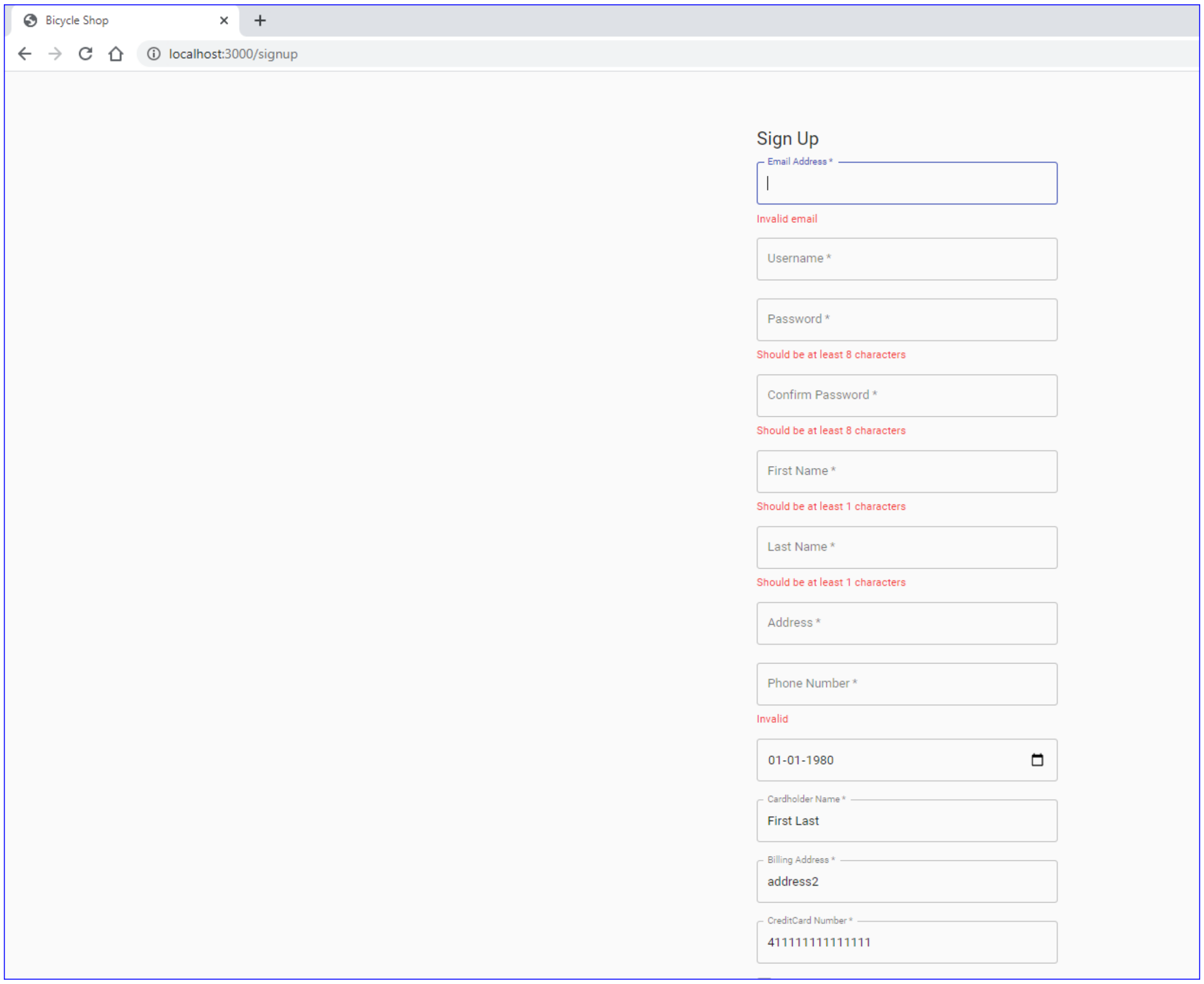
Ludvig Ángeles María
225-123-4567
manager7@bikeshop.com

Rolando Shevaun
225-123-4567
manager8@bikeshop.com

Juantxo Ampelio
225-123-4567
manager9@bikeshop.com

Haleigh Aileen
225-123-4567
manager10@bikeshop.com

SIT_ TC03:
Test case description: To verify validation rules for all the mandatory fields on Sign Up page.
Prerequisite : Bike App website should be correctly configured and running.
Input: Blank/Incorrect user details for signing up
Condition or function under test: .post('/api/accounts/signup')
Expected Output: Error message should be displayed for all the mandatory fields.
Actual Output: User is successfully signed up to the Bike App website.
Test Data: Blank data in input fields for mandatory fields.
Test Result:



Sign Up

Email Address *

Invalid email

Username *

Password *

Should be at least 8 characters

Confirm Password *

Should be at least 8 characters

First Name *

Should be at least 1 characters

Last Name *

Should be at least 1 characters

Address *

Phone Number *

Invalid

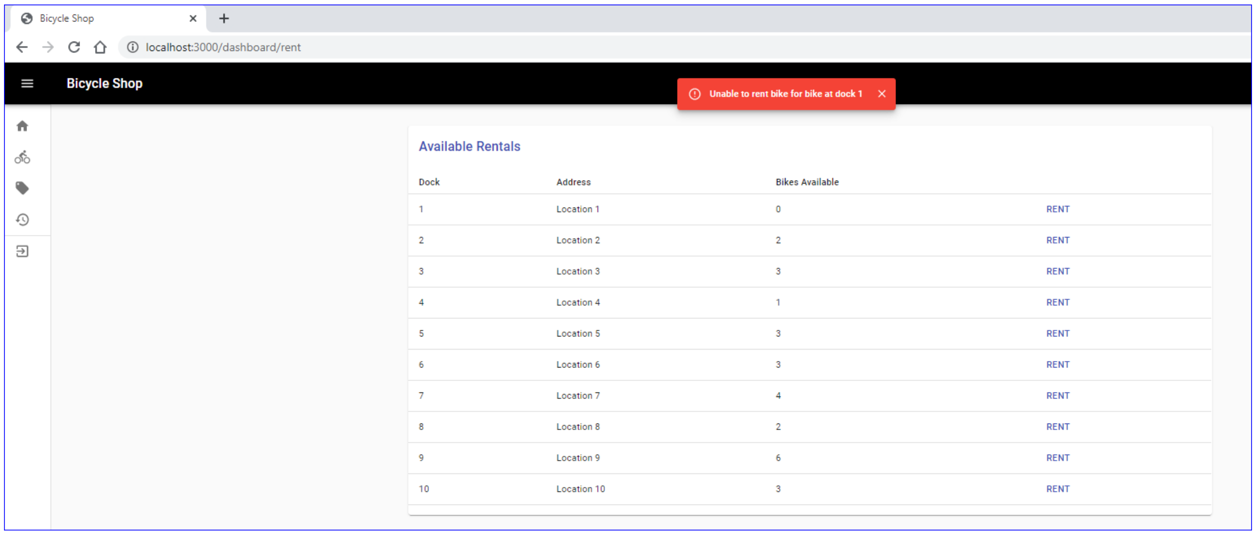
01-01-1980

Cardholder Name *: First Last

Billing Address *: address2

CreditCard Number *: 4111111111111111

SIT_ TC04:
Test case description: To verify validation rules for renting bike when dock station has zero bikes.
Prerequisite : Bike Dock station should have no bikes available for renting. .
Input: Renting Bike from Dock Station 1
Expected Output: Error message should be displayed to the user for not allowing them to rent the bike.
Actual Output: Error message is displayed “Unable to rent bike for bike at dock1.”
Test Result:


Bicycle Shop

localhost:3000/dashboard/rent

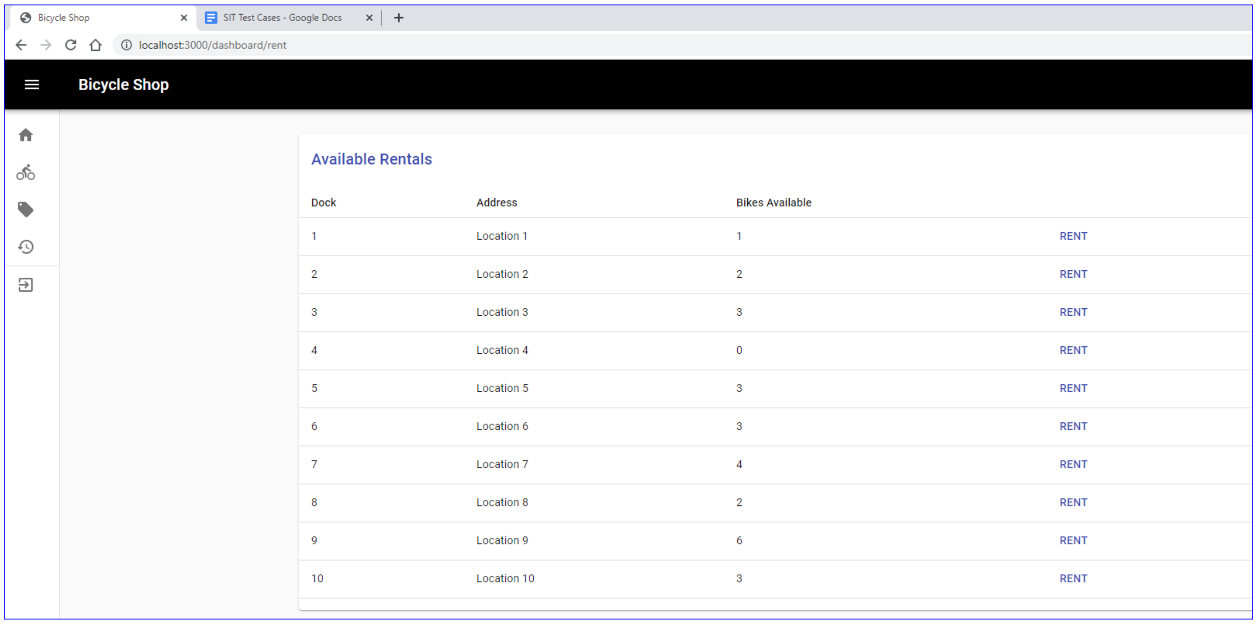
Bicycle Shop

Unable to rent bike for bike at dock 1

Available Rentals

Dock	Address	Bikes Available	
1	Location 1	0	RENT
2	Location 2	2	RENT
3	Location 3	3	RENT
4	Location 4	1	RENT
5	Location 5	3	RENT
6	Location 6	3	RENT
7	Location 7	4	RENT
8	Location 8	2	RENT
9	Location 9	6	RENT
10	Location 10	3	RENT

SIT_ TC05:
Test case description: To verify validation rules for returning bikes when the dock station has reached the maximum limit of bikes at the dock station.
Prerequisite : Bike App website should be correctly configured and running.
Input: Return Bike at Dock Station Location 9.
Expected Output: Error message should be displayed to the user for not allowing them to return the bike.
Actual Output: Error message is displayed “Unable to return bike for transaction xx”
Test Result:


Bicycle Shop

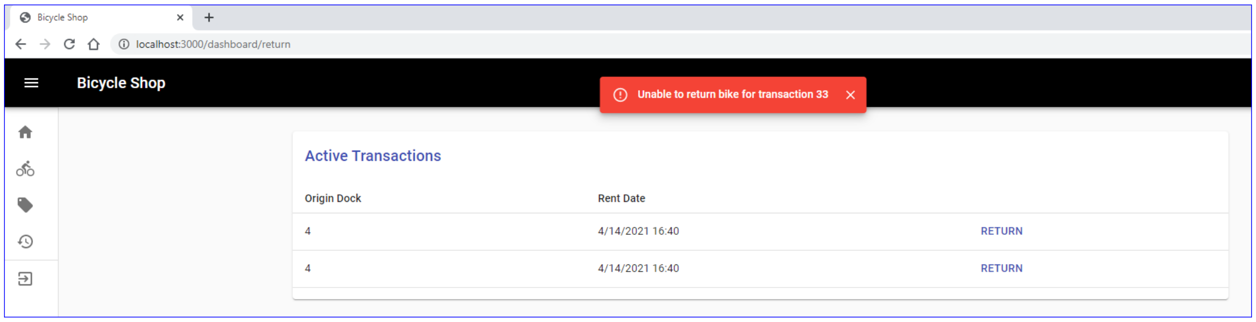
SIT Test Cases - Google Docs

localhost:3000/dashboard/rent

Bicycle Shop

Available Rentals

Dock	Address	Bikes Available	
1	Location 1	1	RENT
2	Location 2	2	RENT
3	Location 3	3	RENT
4	Location 4	0	RENT
5	Location 5	3	RENT
6	Location 6	3	RENT
7	Location 7	4	RENT
8	Location 8	2	RENT
9	Location 9	6	RENT
10	Location 10	3	RENT


Bicycle Shop

localhost:3000/dashboard/return

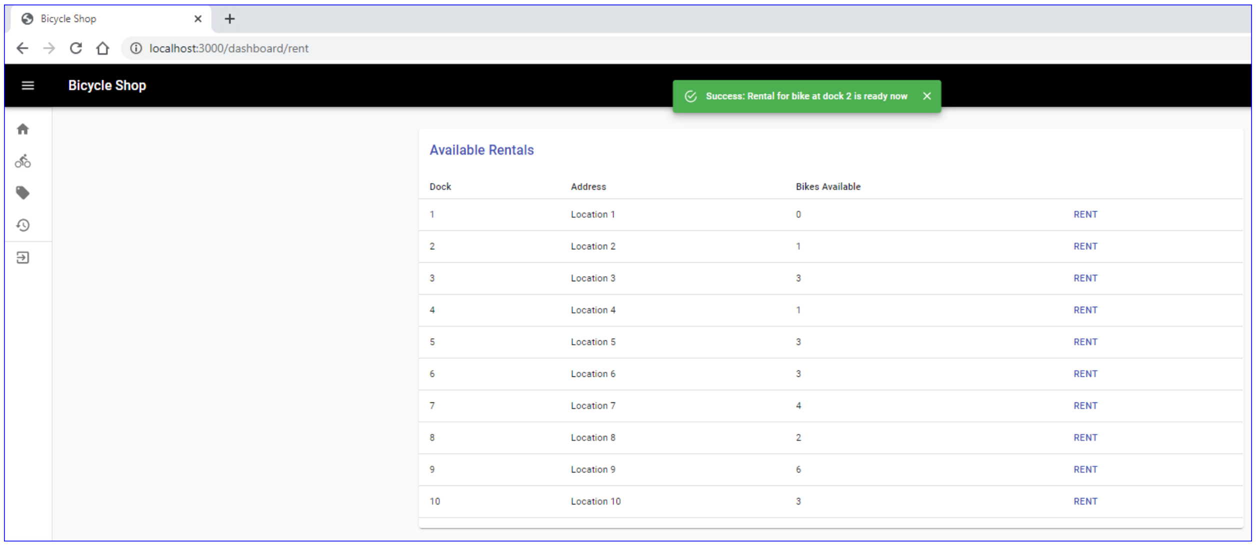
Bicycle Shop

Unable to return bike for transaction 33

Active Transactions

Origin Dock	Rent Date	
4	4/14/2021 16:40	RETURN
4	4/14/2021 16:40	RETURN

SIT_ TC06:
Test case description: To verify the bike renting transaction for the user.
Prerequisite : Bike App website should be correctly configured and running.
Input: Rent Bike from Dock Station 2, and return the bike to available dock station
Expected Output: Transaction should be completed successfully and fees should be applicable as per the consumed time.
$25 should be charged is the bike is not returned to the origin dock station
Actual Output: Transaction completed successfully and the rental fees is charged as per the usage of the bike.
Test Result:
Bike successfully rented from Dock Station 2:


Bicycle Shop

localhost:3000/dashboard/rent

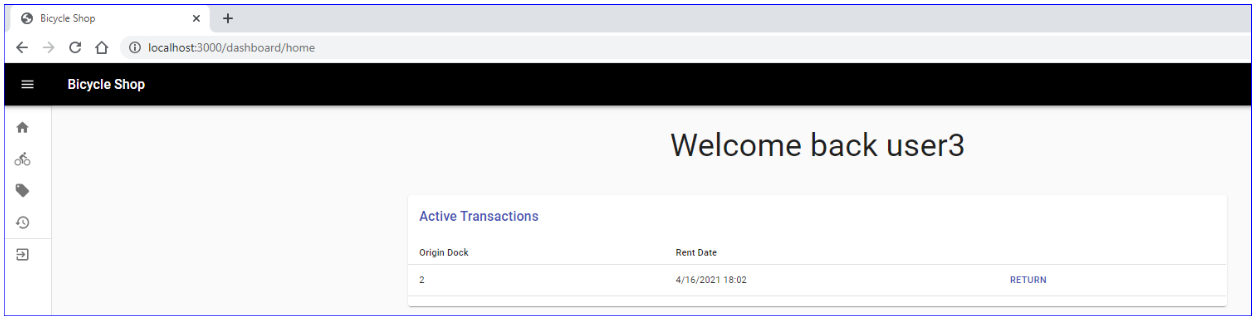
Bicycle Shop

Success: Rental for bike at dock 2 is ready now

Available Rentals

Dock	Address	Bikes Available	
1	Location 1	0	RENT
2	Location 2	1	RENT
3	Location 3	3	RENT
4	Location 4	1	RENT
5	Location 5	3	RENT
6	Location 6	3	RENT
7	Location 7	4	RENT
8	Location 8	2	RENT
9	Location 9	6	RENT
10	Location 10	3	RENT

Active Transaction is displayed for respective user:

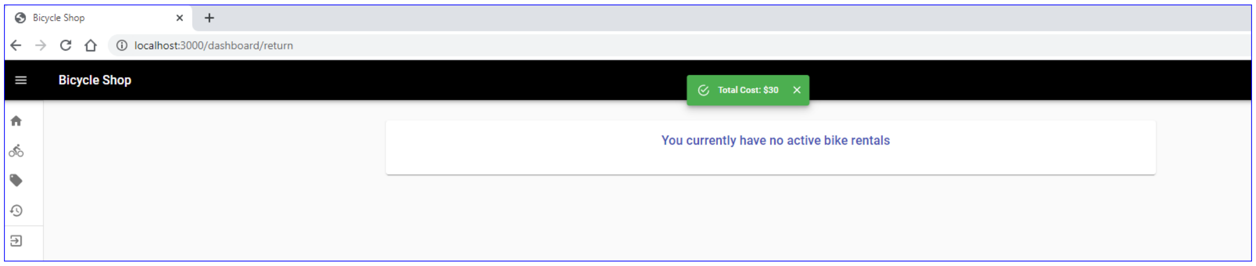


Welcome back user3

Active Transactions

Origin Dock	Rent Date	
2	4/16/2021 18:02	RETURN

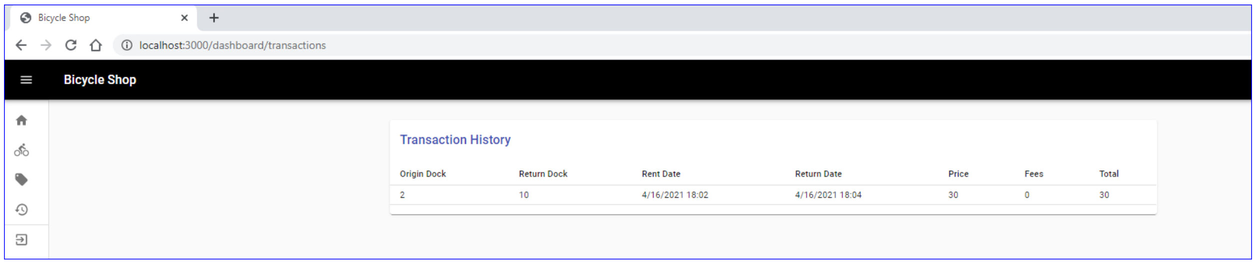
Bike successfully returned.
$30 was charged as the Bike was returned to a different dock station.



Total Cost: $30

You currently have no active bike rentals

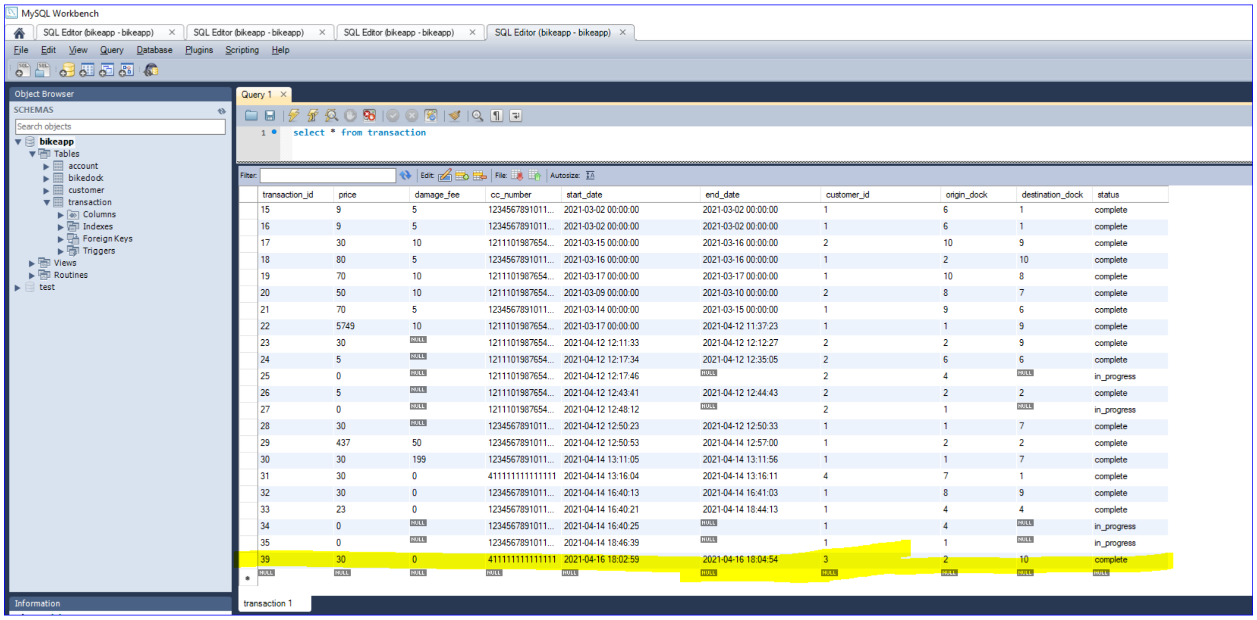
The Transaction history table is updated successfully.



Transaction History

Origin Dock	Return Dock	Rent Date	Return Date	Price	Fees	Total
2	10	4/16/2021 18:02	4/16/2021 18:04	30	0	30

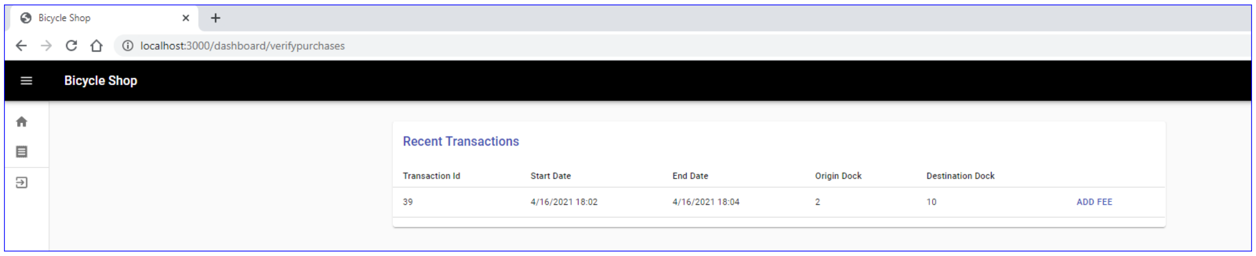
Database is updated successfully :



```
select * from transaction
```

transaction_id	price	damage_fee	cc_number	start_date	end_date	customer_id	origin_dock	destination_dock	status
15	9	5	1234567891011...	2021-03-02 00:00:00	2021-03-02 00:00:00	1	6	1	complete
16	9	5	1234567891011...	2021-03-02 00:00:00	2021-03-02 00:00:00	1	6	1	complete
17	30	10	1211101987654...	2021-03-15 00:00:00	2021-03-16 00:00:00	2	10	9	complete
18	80	5	1234567891011...	2021-03-16 00:00:00	2021-03-16 00:00:00	1	2	10	complete
19	70	10	1211101987654...	2021-03-17 00:00:00	2021-03-17 00:00:00	1	10	8	complete
20	50	10	1211101987654...	2021-03-09 00:00:00	2021-03-10 00:00:00	2	8	7	complete
21	70	5	1234567891011...	2021-03-14 00:00:00	2021-03-15 00:00:00	1	9	6	complete
22	5749	10	1211101987654...	2021-03-17 00:00:00	2021-04-12 11:37:23	1	1	9	complete
23	30	NULL	1211101987654...	2021-04-12 12:11:33	2021-04-12 12:12:27	2	2	9	complete
24	5	NULL	1211101987654...	2021-04-12 12:17:34	2021-04-12 12:35:05	2	6	6	complete
25	0	NULL	1211101987654...	2021-04-12 12:17:46	NULL	2	4	NULL	in_progress
26	5	NULL	1211101987654...	2021-04-12 12:43:41	2021-04-12 12:44:43	2	2	2	complete
27	0	NULL	1211101987654...	2021-04-12 12:48:12	NULL	2	1	NULL	in_progress
28	30	NULL	1234567891011...	2021-04-12 12:50:23	2021-04-12 12:50:33	1	1	7	complete
29	437	50	1234567891011...	2021-04-12 12:50:53	2021-04-14 12:57:00	1	2	2	complete
30	30	199	1234567891011...	2021-04-14 13:11:05	2021-04-14 13:11:56	1	1	7	complete
31	30	0	4111111111111111	2021-04-14 13:16:04	2021-04-14 13:16:11	4	7	1	complete
32	30	0	1234567891011...	2021-04-14 16:40:13	2021-04-14 16:41:03	1	8	9	complete
33	23	0	1234567891011...	2021-04-14 16:40:21	2021-04-14 18:44:13	1	4	4	complete
34	0	NULL	1234567891011...	2021-04-14 16:40:25	NULL	1	4	NULL	in_progress
35	0	NULL	1234567891011...	2021-04-14 18:46:39	NULL	1	1	NULL	in_progress
39	30	0	4111111111111111	2021-04-16 18:02:59	2021-04-16 18:04:54	3	2	10	complete
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

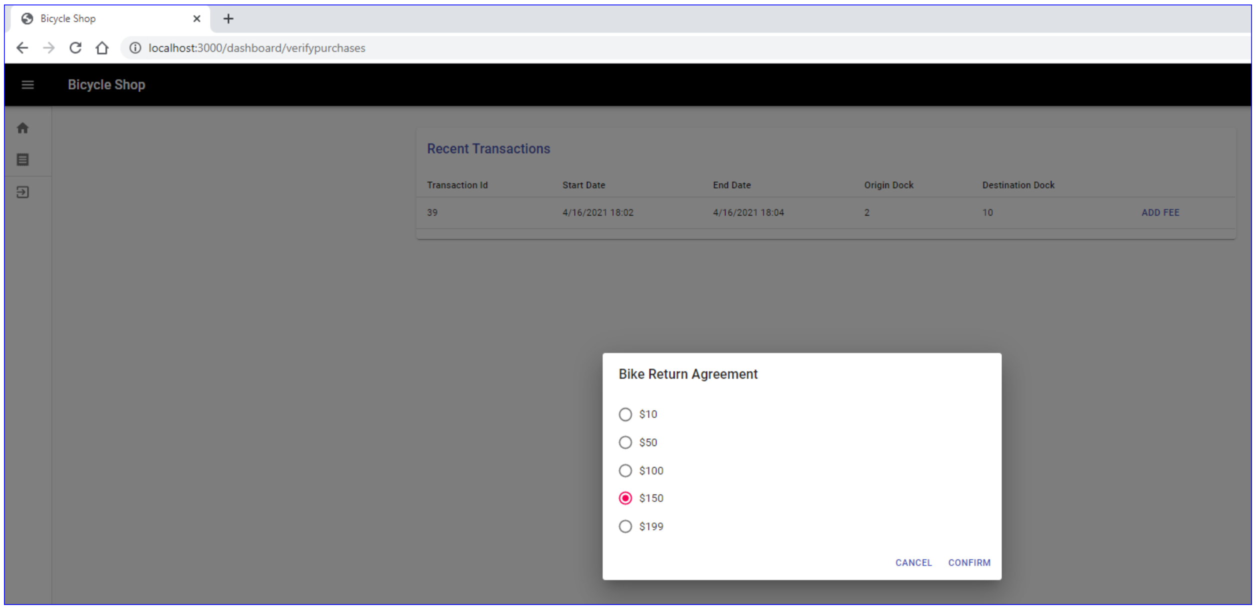
SIT_ TC07:
Test case description: To verify the damage fee transaction.
Prerequisite : SIT_TC06 should be completed.
Input: Manager will charge a damage fee to the credit card of the completed transaction.
Expected Output:
Actual Output:
Test Result:
Active transaction is displayed for Manager login:



Recent Transactions

Transaction Id	Start Date	End Date	Origin Dock	Destination Dock	
39	4/16/2021 18:02	4/16/2021 18:04	2	10	ADD FEE

Add Fee menu is successfully displayed for Manager Role:



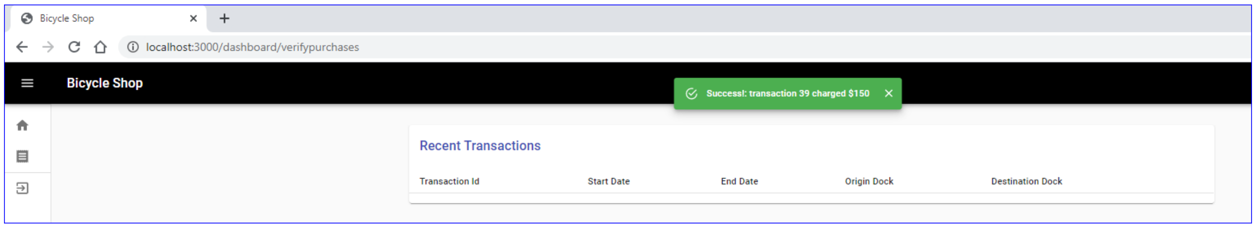
Recent Transactions

Transaction Id	Start Date	End Date	Origin Dock	Destination Dock	
39	4/16/2021 18:02	4/16/2021 18:04	2	10	ADD FEE

Bike Return Agreement

$10 $50 $100 [✓ $150] $199

Confirmation message displayed on confirming the 'Damage Fee' amount:

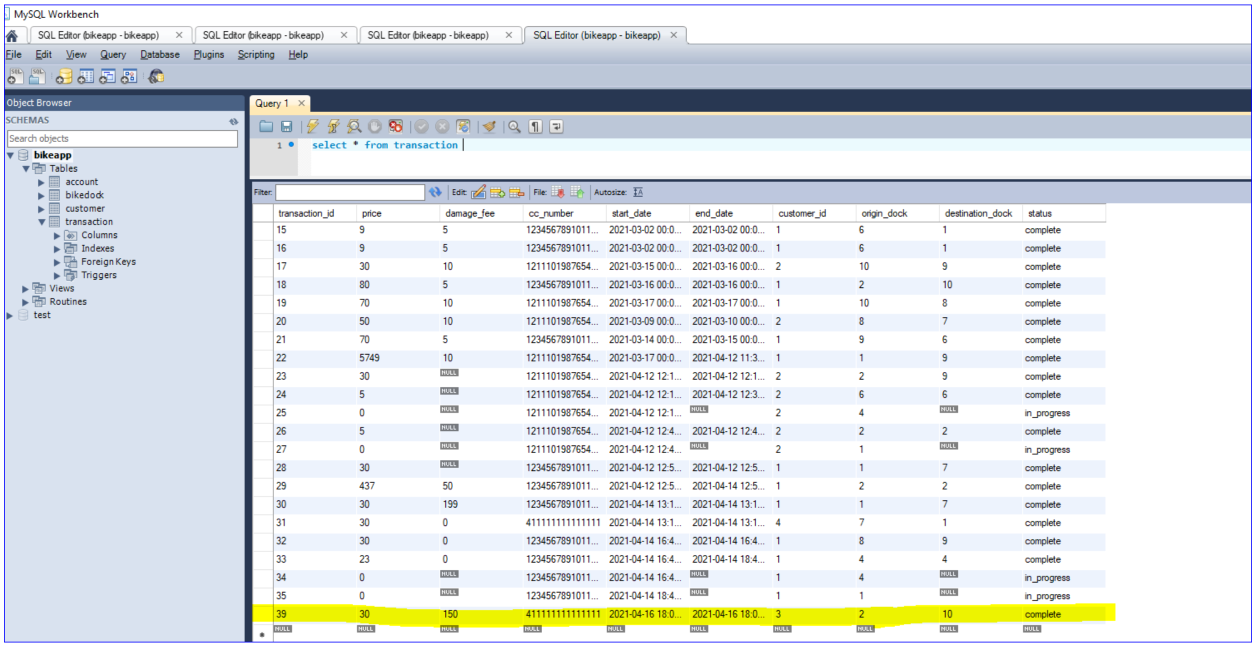


Success!: transaction 39 charged $150

Recent Transactions

Transaction Id	Start Date	End Date	Origin Dock	Destination Dock

Damage Fee is successfully displayed in Database Transaction Report:



```
select * from transaction
```

transaction_id	price	damage_fee	cc_number	start_date	end_date	customer_id	origin_dock	destination_dock	status
15	9	5	1234567891011...	2021-03-02 00:0...	2021-03-02 00:0...	1	6	1	complete
16	9	5	1234567891011...	2021-03-02 00:0...	2021-03-02 00:0...	1	6	1	complete
17	30	10	1211101987654...	2021-03-15 00:0...	2021-03-16 00:0...	2	10	9	complete
18	80	5	1234567891011...	2021-03-16 00:0...	2021-03-16 00:0...	1	2	10	complete
19	70	10	1211101987654...	2021-03-17 00:0...	2021-03-17 00:0...	1	10	8	complete
20	50	10	1211101987654...	2021-03-09 00:0...	2021-03-10 00:0...	2	8	7	complete
21	70	5	1234567891011...	2021-03-14 00:0...	2021-03-15 00:0...	1	9	6	complete
22	5749	10	1211101987654...	2021-03-17 00:0...	2021-04-12 11:3...	1	1	9	complete
23	30	NULL	1211101987654...	2021-04-12 12:1...	2021-04-12 12:1...	2	2	9	complete
24	5	NULL	1211101987654...	2021-04-12 12:1...	2021-04-12 12:3...	2	6	6	complete
25	0	NULL	1211101987654...	2021-04-12 12:1...	NULL	2	4	NULL	in_progress
26	5	NULL	1211101987654...	2021-04-12 12:4...	2021-04-12 12:4...	2	2	2	complete
27	0	NULL	1211101987654...	2021-04-12 12:4...	NULL	2	1	NULL	in_progress
28	30	NULL	1234567891011...	2021-04-12 12:5...	2021-04-12 12:5...	1	1	7	complete
29	437	50	1234567891011...	2021-04-12 12:5...	2021-04-14 12:5...	1	2	2	complete
30	30	199	1234567891011...	2021-04-14 13:1...	2021-04-14 13:1...	1	1	7	complete
31	30	0	4111111111111111	2021-04-14 13:1...	2021-04-14 13:1...	4	7	1	complete
32	30	0	1234567891011...	2021-04-14 16:4...	2021-04-14 16:4...	1	8	9	complete
33	23	0	1234567891011...	2021-04-14 16:4...	2021-04-14 18:4...	1	4	4	complete
34	0	NULL	1234567891011...	2021-04-14 16:4...	NULL	1	4	NULL	in_progress
35	0	NULL	1234567891011...	2021-04-14 18:4...	NULL	1	1	NULL	in_progress
39	30	150	4111111111111111	2021-04-16 18:0...	2021-04-16 18:0...	3	2	10	complete
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

8.2 Defect Report:

Defect ID	**Defect Name**	**Priority**	**Severity**	**Defect Description**	**Status**
D001	Error in calculating total_rental s	High	Medium	The number of rentals is not accurate at the dock station. Expected Result: 20 Actual Result:0	Resolved

8. Assumptions

- The manager will decide the damage and extra fees(if applicable) upon returning the bike to the dock station.
- There is no lower limit for the availability of bikes at the dock station.

9. Terms/Acronyms

List of terms/acronyms used in the test plan or project.

Term/Acronym	**Definition**
UC	Use Case
TC	Test Case
api	Application Programming Interface
SIT	System Integration Test

Backlog for the first sprint

1. Database design, and creation
2. Methods for user account creation and account login (minus validation)
3. Methods for manager account creation and login
4. Webpages for signup.
    - 4.1. UI Design for pages.

Backlog for the second sprint

1. Account Creation/Login Web pages
    - 1.1. Store user credentials
        - 1.1.1. Address
        - 1.1.2. Birthdate
        - 1.1.3. Phone number
        - 1.1.4. Credit card information
        - 1.1.5. Email address
        - 1.1.6. Validate credit card and email address
        - 1.1.7. Validate user login
2. Bike Rental Rates Calculator
    - 2.1. Calculates rental fee when returned
        - 2.1.1. $5 per half hour
        - 2.1.2. $9 per hour, portion of hour used charged full price of hour

2.1.3. $25 if returned to different dock
2.1.4. $500 if bike is not returned
2.1.5. $200 if bike is damaged depending on damage

3. Docks/Bikes Webpage
    3.1. Uniquely identify each of the 10 docks
        3.1.1. Track number of bikes at each dock, not exceeding 6 bikes per dock

Create User view and fully complete the basic rent-return bike transaction through the website.

Backlog for the third sprint

6. Generate Weekly Report
    1.1 Show total bike usage
    1.2 Calculate income of each dock

Contributions

Every member of the team worked together equally to create this document.