System Design Document for the RhodeIT Mobile Application

Version 1.0

Siphamandla Mjoli

G14M1190

1. Introduction

A Software design document is a document that provides documentation which will aid in the software development phase of the system, by providing details as to how the proposed system should be built. The contents of a Software Design document include narrative and graphical documentation of the design of the proposed system such as use case models, hierarchy diagrams, and activity diagrams, data flow diagrams and any other information necessary for the development of the system.

* 1. Purpose

The purpose of this document is to provide a description of the design of the proposed system. This will allow for a better understanding of the system as a whole and will allow for the software development phase to proceed with a clear understanding as to what is to be built.

* 1. Scope

This document serves as a base level system which will work as a proof of concept for building a system that provides base functionality and feasibility of the different aspects of the system.

* 1. Overview

This document will be divided into various sections which may also contain subsections these sections can be listed as follows

* Introduction
* Use Cases
* Use Case Narratives
* Design Overview
* Mind Map
* Activity Hierarchy Diagram
* Flow Chart
* Class diagram

1. Use Cases
   1. Actors
      1. Student

A Student is user who generates allot of activity on the system

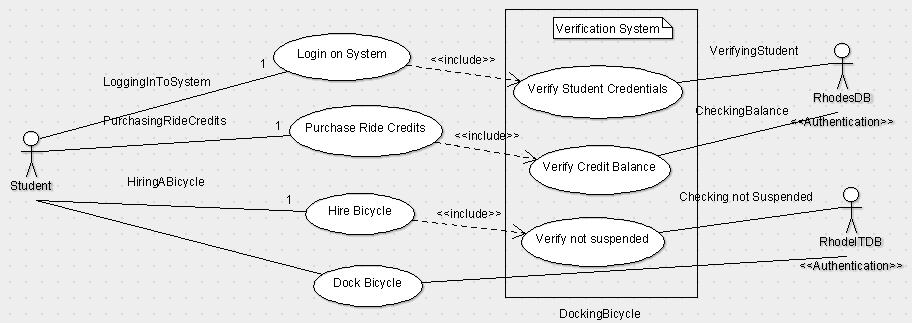
* 1. Administrative User

The administrative user is a user who is responsible for overseeing and managing the system as a whole.

* 1. Use Case Scenarios for Actors
     1. Student Use Case Scenario

The proposed use cases for the Student Actor are as follows

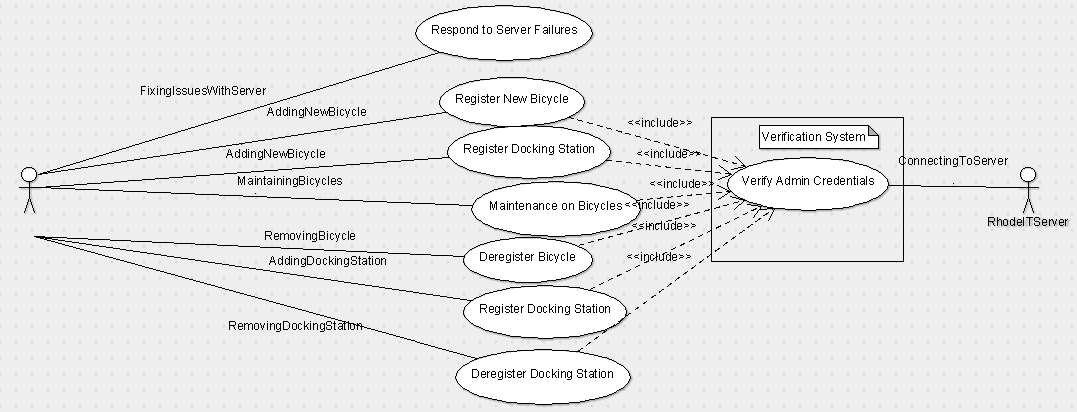
* Registration on the system
* Purchasing Ride Credits on system
* Hiring a Bicycle
* Docking a Bicycle
  + - 1. Use Case Diagrams
         1. Use Case Diagram for a Student



* + 1. Use Case Scenario for Administrative User

The proposed use cases for the Student Actor are as follows

* Respond to server failures
* Register New Bicycles
* Maintenance on Bicycles
* Deregister Bicycle
* Register Docking Station
* Deregister Docking Station
  + - 1. Use Case Diagram for Administrative User



* + 1. Use Case Narratives

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RhodeIT | | | | | | |
| **Name of use case:** | | | | Registration on the system | | |
| Purpose: | | | | This use case describes how a Student will login to the RhodeIT Mobile Application | | |
| Overview: | | | | The use case will look at the various processes involved in the registration of a student’s credentials | | |
| Actors: | | | | Student | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented. | | |
| Preconditions: | | | | Students Mobile Phone must have an internet connection | | |
| Postconditions: | | | | Student is registered on system | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | When the user launches the RhodeIT Mobile Application for the first time | | | | 2. | The system will display a login screen |
|  |
| 3. | The student will then enter they student login credentials provided by the university and click login | | | | 4. |  |
| * If there aren’t any empty fields the system will then initiate the login process * If the credentials exists on the Rhodes University’s Database student is registered on Platform * Register Student With Smart Contract and Store Transaction Receipt |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * If the student has any empty fields the system will respond with an error about the field that is empty * Warn the user if there isn’t any internet connection * If the credentials don’t exist on the Rhodes Database an error will returned to user | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Student Login Use Case Diagram | | |
| System Operations: | | | | FirstRun()  Process()  hasLoggedInBefore()  setUpView()  VerifyStudent()  RegisterStudentWithSmartContract()  StoreTransactionReciept()  Login() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RhodeIT | | | | | | |
| **Name of use case:** | | | | Purchasing Ride Credits on system | | |
| Purpose: | | | | The use case describes the process of purchasing Ride Credits | | |
| Overview: | | | | The use case examines the different processes involved when a student wishes to purchases Ride credits on the platform | | |
| Actors: | | | | Student | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented. | | |
| Preconditions: | | | | Student is registered | | |
| Postconditions: | | | | Student Ride account is credited with x amount | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | When a user clicks on the Purchase button | | | | 2. | System opens up popup box which has information about the amount the Student currently has as well as the maximum amount a Student can purchase and a input box which can be used to enter a desired amount to purchase |
|  |
| 3. | The student enters an amount to purchase and clicks purchase | | | | 4. | The system will communicate with the Rhodes server about the student wishing to purchase Ride credits and update the students account accordingly |
| Sync Balance with Smart Contract and store transaction Receipt |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * If the student has any empty fields the system will respond with an error about the field that is empty * Warn the user if there isn’t any internet connection | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | StudentLogin Use Case Diagram | | |
| System Operations: | | | | Purchase()  UpdateStudentRideBalance()  SyncBalanceWithSmartContract()  StoreTransactionReciept() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RhodeIT | | | | | | |
| **Name of use case:** | | | | Hiring a Bicycle | | |
| Purpose: | | | | The use case describes the process Hiring a Bicycle | | |
| Overview: | | | | The use case examines the different processes involved when a student wishes to Hire a Bicycle on the platform | | |
| Actors: | | | | Student | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | Student has enough Ride Credits | | |
| Postconditions: | | | | Student is leased a Bicycle | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | Student clicks on a Docking stations location on the map | | | | 2. | The System will display all available bicycles on the Platform on a slide up menu list |
|  |
| 3. | The Student clicks on the Hire Bicycle on any of the available bicycles on the list | | | | 4. | System checks if the user is not banned and proceeds to assign an unlock code and communicates this with the RhodeIT server |
|  |
| 5. | Student removes lock on bicycle | | | | 6. | Lock auto locks and store transaction receipt |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * If student is banned, student will not be allowed to hire a bicycle * System fails to unlock bicycle due to communication error (network issues) | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Student Login Use Case Diagram | | |
| System Operations: | | | | GetAvailableBicyclesOnDockingStation()  VerifyNotBanned()  GenerateHireCode()  UnlockBicycle()  SyncTransactionWithSmartContract()  StoreTransactionReciept() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RhodeIT | | | | | | |
| **Name of use case:** | | | | Docking a bicycle | | |
| Purpose: | | | | The use case describes the process Hiring a Bicycle | | |
| Overview: | | | | The use case examines the different processes involved when a student wishes to Hire a Bicycle on the platform | | |
| Actors: | | | | Student | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | Student hired a Bicycle | | |
| Postconditions: | | | | Bicycle is locked in Docking Station | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | When the Student clicks on the Dock bicycle button | | | | 2. | System will provide a list of available empty slots on docking station |
|  |
| 3. | Student will then click on dock here button | | | | 4. | System will allocate the free spot on the docking station to user for 30 minutes and notify the user |
|  |
| 5. | Student clicks on dock button | | | | 6. | System uses the unlock key allocated to Student when they hired the bicycle to unlock the specific lock |
|  |
| 7. | Student locks bicycle using lock | | | | 8. | System notifies user about successful locking of bicycle  Sync Transaction with Smart Contract marking bicycle as being returned |
| Store Transaction Receipt |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * Student fails to dock station in the given 30 minutes * Lock doesn’t lock successfully * Unlock key doesn’t unlock lock | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Student Use Case Diagram | | |
| System Operations: | | | | GetAvailableDockingSpots()  AllocateSpot()  UnlockLock()  SyncTransactionWithSmartContract()  StoreTransactionReciept()  NotifyDockingStatus() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RHODEIT | | | | | | |
| **Name of use case:** | | | | **Respond to Server Failures** | | |
| Purpose: | | | | The use case describes the process of fixing an issues with the server | | |
| Overview: | | | | The use case examines the different processes involved when the Administrative user has to respond to server issues | | |
| Actors: | | | | Administrative user | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | The server is unreachable from mobile app | | |
| Postconditions: | | | | Server issues fixed | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | Identify what caused the server to not be reachable | | | | 2. |  |
|  |
| 3. | Fix server issue accordingly | | | | 4. |  |
|  |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | Unable to determine issue with server | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Administrative Use case diagram | | |
| System Operations: | | | |  | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RHODEIT | | | | | | |
| **Name of use case:** | | | | **Register New Bicycle** | | |
| Purpose: | | | | The use case describes the process of Registering a new Bicycle | | |
| Overview: | | | | The use case examines the different processes involved when the Administrative user has to Register a new Bicycle | | |
| Actors: | | | | Administrative user | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | A new Bicycle has to be added | | |
| Postconditions: | | | | Bicycle successfully added | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | Check if a new bicycle can be added | | | | 2. | Check on all docking stations if a bicycle slot exist or if there are issues with any bicycle and return list with information about bicycle |
|  |
| 3. | Select docking station from list of bicycle with issues and register bicycle on that docking station | | | | 4. | Add bicycle on specified docking station and mark previous bicycle as to be removed and return success |
| Register Bicycle on Smart Contract  Store Transaction Receipt |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * No spot to add new bicycle * Adding bicycle to docking station failure (server issues) | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Administrative Use case diagram | | |
| System Operations: | | | | CanAddBicycle()  AddBicycle()  RegisterBicycleWithSmartContract()  StoreTransactionReciept() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RHODEIT | | | | | | |
| **Name of use case:** | | | | **Register Docking Station** | | |
| Purpose: | | | | The use case describes the process of Registering a new Docking station on Campus | | |
| Overview: | | | | The use case examines the different processes involved when the Administrative user has to Register a new Docking station on Campus | | |
| Actors: | | | | Administrative user | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | Docking station is not already registered | | |
| Postconditions: | | | | Docking Station successfully registered | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | Check if a new Docking station can be added | | | | 2. | Iterate through all docking station comparing them with the proposed new docking station to be added respond with the appropriate status of the search |
|  |
| 3. | Add new docking station | | | | 4. | Add docking station on RhodeIT server  Respond with appropriate message |
| Register Docking Station on Smart Contract  Store Transaction Receipt |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * Docking station already exist on Platform * Adding docking to docking station failure (server issues) | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Administrative Use case diagram | | |
| System Operations: | | | | DockingStaionExists()  RegisterDockignStation()  RegisterDockingStationWithSmartContract()  StoreTransactionReciept() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RHODEIT | | | | | | |
| **Name of use case:** | | | | **Maintenance on Bicycles** | | |
| Purpose: | | | | The use case describes the process of conducting Maintenance on the Bicycles registered on the Platform | | |
| Overview: | | | | The use case examines the different processes involved when the Administrative user has to conduct Maintenance on the Bicycles registered on the Platform | | |
| Actors: | | | | Administrative user | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | Registered Bicycles on are marked as “Require Maintenance” | | |
| Postconditions: | | | | Bicycle setup for Maintenance | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | Check if any bicycles on system have been reported as “Require Maintenance” | | | | 2. | Iterate through all docking station Checking if any of the bicycles have been marked as needing Maintenance and return the list to user |
|  |
| 3. | Temporarily deregister bicycles from Docking station | | | | 4. | Deregister bicycles on platform temporarily and respond with the appropriate message |
| Store Transaction Receipt |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * Deregistration failure (server issues) | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Administrative Use case diagram | | |
| System Operations: | | | | RequireMaintainanceBicycle()  DeRegisterTemp()  DeregisterBicycleTempOnSmartContract()  StoreTransactionReciept() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

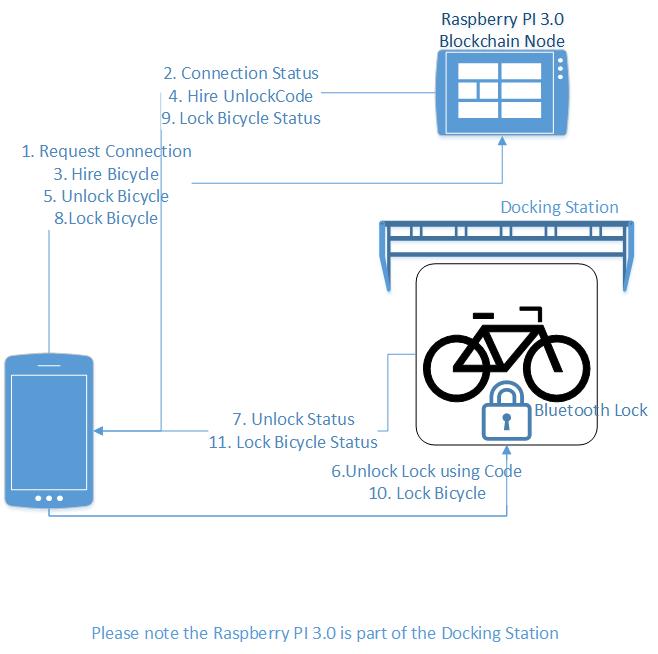
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RHODEIT | | | | | | |
| **Name of use case:** | | | | **Deregister Bicycle** | | |
| Purpose: | | | | The use case describes the process of Deregistering a Bicycle | | |
| Overview: | | | | The use case examines the different processes involved when the Administrative user has to Deregister a Bicycle | | |
| Actors: | | | | Administrative user | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | Bicycle is Registered on System | | |
| Postconditions: | | | | Bicycle removed | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | Check if bicycle is registered on system | | | | 2. | Iterate through all docking station Checking if any of the bicycle is registered and respond with the appropriate results |
|  |
| 3. | deregister bicycle from Docking station | | | | 4. | Deregister bicycle on platform and respond with the appropriate message |
| Store Transaction Receipt |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * Deregistration failure (server issues) | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Administrative Use case diagram | | |
| System Operations: | | | | IsRegisteredBicycle()  DeRegisterBicycle()  RegisterBicycleWithSmartContract()  StoreTransactionReciept() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Expanded Use case Narrative: RHODEIT | | | | | | |
| **Name of use case:** | | | | **Deregister Docking Station** | | |
| Purpose: | | | | The use case describes the process of Deregistering a Docking Station | | |
| Overview: | | | | The use case examines the different processes involved when the Administrative user has to Deregister a Docking Station | | |
| Actors: | | | | Administrative user | | |
| Type: | | | | * Essential – the use case narrative ignores the specifics of the technology with which the system is or will be implemented | | |
| Preconditions: | | | | Docking Station is Registered on System | | |
| Postconditions: | | | | Docking Station removed | | |
| Special requirements: | | | |  | | |
| **Main Flow of Events** | | | | | | |
| **ACTOR ACTION** | | | | | **SYSTEM RESPONSE** | |
| 1. | Check if Docking Station is registered on system | | | | 2. | Iterate through all docking station Checking if any match the provided Docking Station and respond with the appropriate results |
|  |
| 3. | deregister Docking Station | | | | 4. | Deregister Docking Station and respond with the appropriate message |
| Store Transaction Receipt |
| 5. |  | | | | 6. |  |
|  |
| 7. |  | | | | 8. |  |
|  |
| 9. |  | | | | 10. |  |
|  |
|  |  | | | |  |  |
| **Alternative Flow of Events** | | | | | | |
| Line # | |  | * Deregistration failure (server issues) | | | |
|  | |  |  | | | |
| Activity Diagrams: | | | | Administrative Use case diagram | | |
| System Operations: | | | | IsRegisteredDockingStation()  DeRegisterDockingStation()  DeRegisterDockingStationOnSmartContract()  StoreTransactionReciept() | | |
|  | | | |  | | |
| Author: | | | | Siphamandla Mjoli | | |
| Date Created: | | | | 26 April 2019 | | |
| Date last modified: | | | | 26 April 2019 | | |
| Modified by: | | | | Siphamandla Mjoli | | |

1. Design Overview

This section main purpose is to introduce and give a brief overview of the design of the system. The System Architecture is a way of providing the overall view of the proposed system by placing it into context with systems outside of its boundary. This aids the reader with quickly grasping the design of the system before reading any further documentation.

* 1. Systems Architecture
     1. Overall Structure of the RhodeIT Bicycle Sharing App

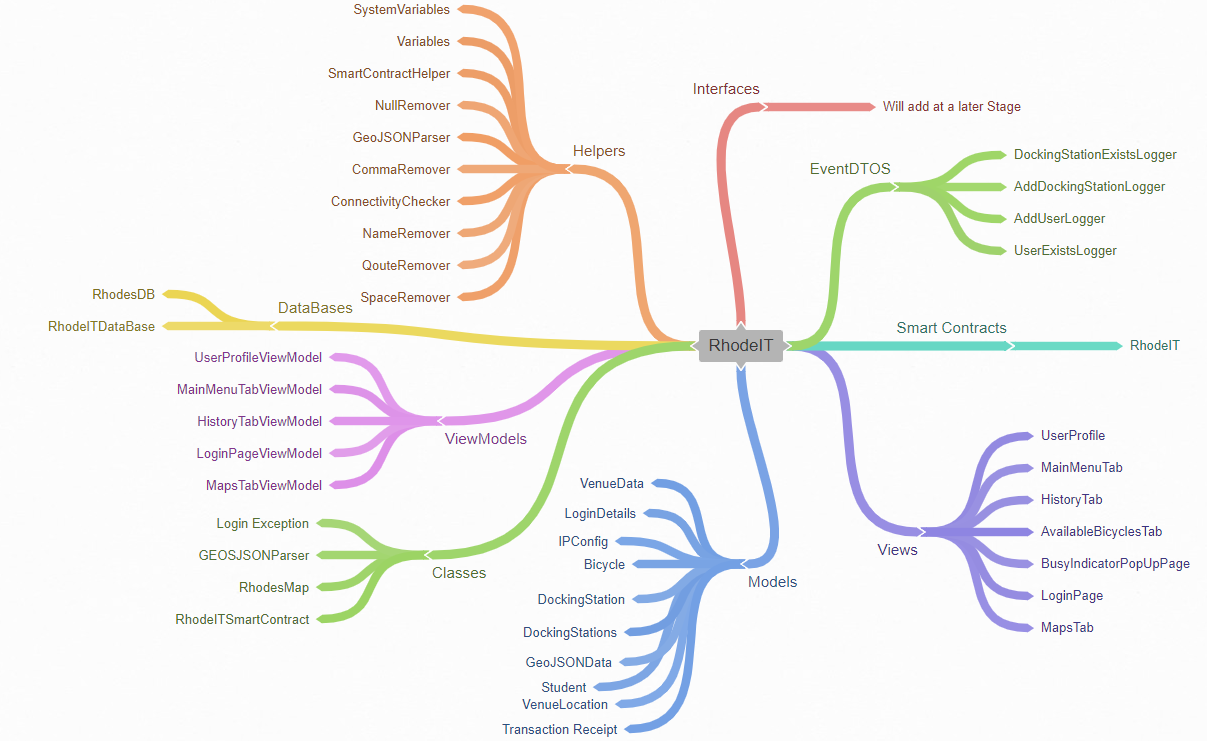


* 1. Mind Map

The aim of this diagram is to group all related objects that are proposed for use on the system, this will help grasp knowledge and comprehend ideas from their counterparts easily and in a visual manner.

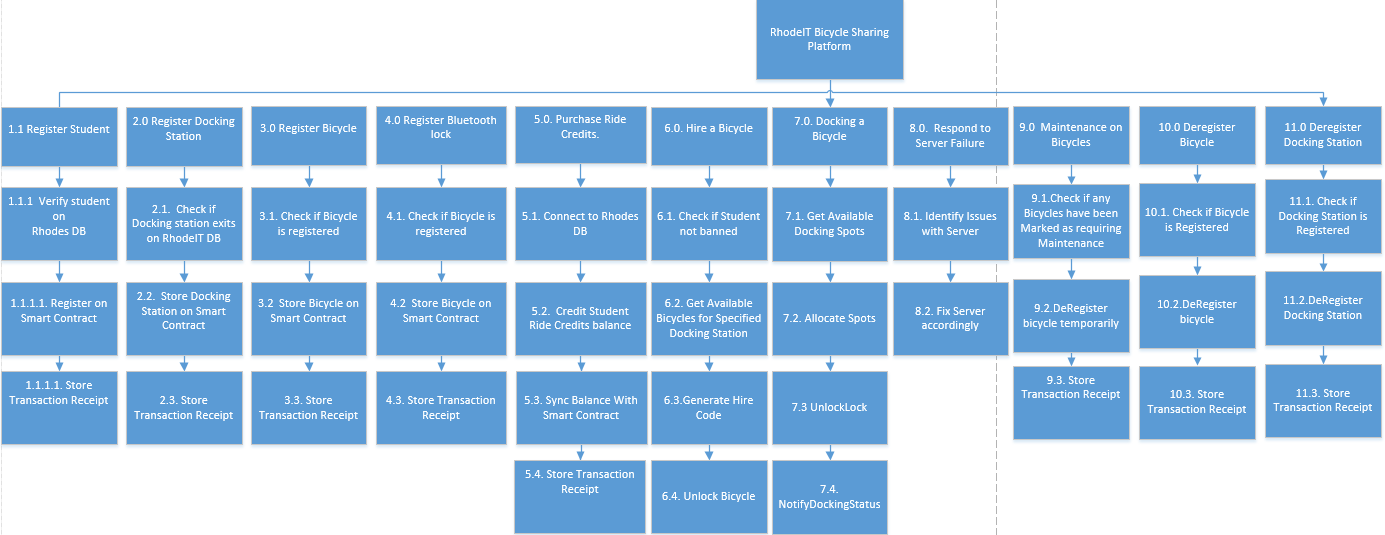
* + 1. Visual Representation Of Proposed Objects For The RhodeIT Mobile Application

+



* 1. Activity Hierarchy Diagram

The Activity Hierarchy Diagram (AHD) identifies the lowest-level processes of interest to the business through decomposition. AHD shows levels of increasing detail for each function and process until activities decompose to the lowest level (elementary processes).

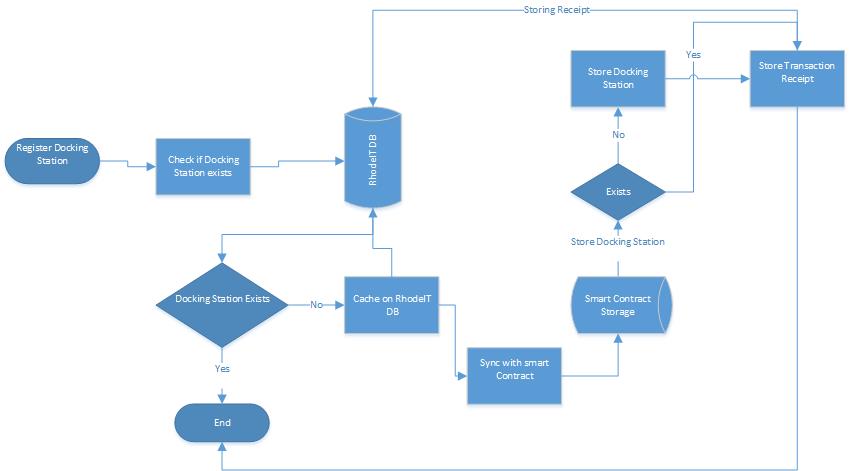
* + 1. Overall Structure of the RhodeIT Mobile Application Activity Hierarchy Diagram
  1. Data Flow Charts

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

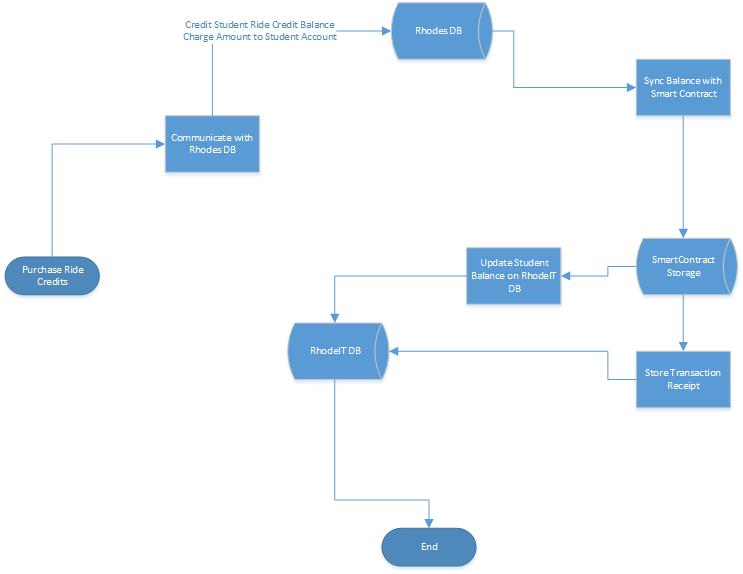
* + 1. User Login Data Flow Diagram



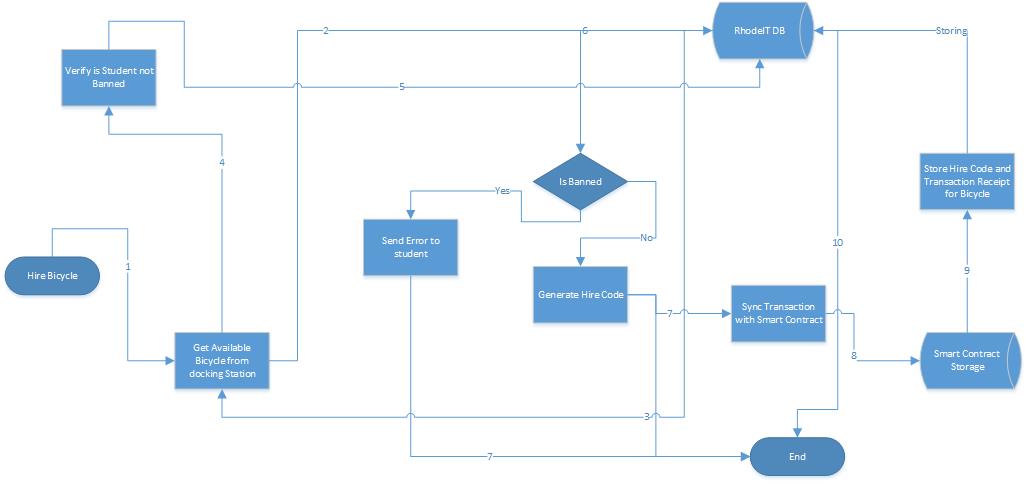
3.4.2. Docking Station Registration Data Flow Diagram



* + 1. Purchase Ride Credits Data Flow Diagram



* + 1. Hire a Bicycle Data Flow Diagram



* + 1. Dock Bicycle Data Flow Diagram

