**System Design Document** **for** **Spacecraft-Control-Center Training and Testing Environment (STaTE)**

Juliana Altamira

Carly Bosma

Jeff Cevallos

Kyle Garber

Jeremy Mog

Jesse Slager

Annamaria Summer

| Version/Author(s) | Date |
| --- | --- |
| 1.0/Jeff Cevallos | 10/4/2022 |
| 1.1/Kyle Garber | 11/7/2022 |
| 1.2/Jeff Cevallos | 11/7/2022 |
| 2.0/Juliana Altamira, Annamaria Summer, Jeff Cevallos, Jesse Slager, Carly Bosma, Jeremy Mog, Kyle Garber | 11/8/2022 |

TABLE OF CONTENT

[**1 Introduction**](#_heading=h.t89dpaeuwpne) **3**

[1.1 Purpose and Scope](#_heading=h.1fob9te) **3**

[**1.2 Project Executive Summary**](#_heading=h.e9csrhski9cd) **3**

[1.2.1 System Overview](#_heading=h.degcfmxvpoma) 3

[1.2.2 Design Constraints](#_heading=h.scr5bhoz67jx) 4

[1.2.3 Future Contingencies](#_heading=h.wa7ig111sy3f) 5

[Missions](#_heading=h.dzrnn3wpmakc) 5

[Browser Compatibility](#_heading=h.1yu1ebj5j5nf) 5

[**1.3 Document Organization**](#_heading=h.vw274kj8o1kf) **6**

[**1.4 Project References**](#_heading=h.w403de4rdr45) **6**

[Reference for Console Information:](#_heading=h.oiff0hs0uz7j) 6

[**1.5 Glossary**](#_heading=h.h036s7nw7r7o) **7**

[**2 System Architecture**](#_heading=h.s4yg265mf0je) **7**

[**2.1 System Hardware Architecture**](#_heading=h.4tpkcqlaq22h) **7**

[**2.2 System Software Architecture**](#_heading=h.85if9cydqpwp) **8**

[**2.3 Internal Communications Architecture**](#_heading=h.du6ydwmt6wep) **9**

[**3 Human-Machine Interface**](#_heading=h.c5einqz59ewm) **10**

[**3.1 Inputs**](#_heading=h.48qqjlz8v4xf) **10**

[**3.2 Outputs**](#_heading=h.cy7upx9iqjsk) **13**

[**4 Detailed Design**](#_heading=h.hjl88mysx470) **14**

[**4.1 Hardware Detailed Design**](#_heading=h.c37sex3nge10) **14**

[**4.2 Software Detailed Design**](#_heading=h.p2gxaep9dbt7) **16**

[STaTE User Classes](#_heading=h.k4gifocsrcco) 16

[Simulation](#_heading=h.jgdgy3nekros) 17

[Website Application](#_heading=h.gxgzn7k8d2nn) 19

[High Level System Overview](#_heading=h.43iqxdxqxhbp) 19

[**4.3 Internal Communications Detailed Design**](#_heading=h.sbns2emfgx6d) **20**

[**5 External Interfaces**](#_heading=h.8jb3wwy4ie26) **20**

[**5.1 Interface Architecture**](#_heading=h.3ph9a08sob0n) **20**

[**5.2 Interface Detailed Design**](#_heading=h.oul9p3ugi8ed) **21**

[**6 System Integrity Controls**](#_heading=h.oeyuaa6q19zv) **22**

## Introduction

## Purpose and Scope

This document describes the high-level overview, system architecture, operating environment, files and database design, input and output formats, detailed design, external interfaces, and system integrity controls for the STaTE project.

## Project Executive Summary

This section provides an overview of the STaTE project from a management perspective, showing the framework with which the system design was conceived.

### **System Overview**

The STaTE project is a platform that allows a Test conductor to provide simulated spacecraft known as SimCraft to be managed by teams of Flight Operators. Test Conductors do this by first creating a SimCraft. To create a SimCraft, the specifications of the SimCraft’s subsystems and the Flight Operators that will manage each SimCraft subsystem must be provided to the system. Once created, the SimCraft runs constantly unless paused or terminated by the Test Conductor. Flight Operator subsystem assignment can be changed dynamically by the Test Conductor while a SimCraft is active. The Test Conductor can manipulate the values of the SimCraft subsystem attributes in ways that require a response from the Flight Operator managing that subsystem. Flight Operators respond to these events through a SimCraft console that displays the allowed actions pertaining to their assigned subsystem. The Test Conductor can view a log maintained by the SimCraft that records and grades individual Flight Operator’s actions and responses. The Test Conductor, Flight Operators, and Maintenance Personnel are all required to log into their respective user accounts to perform the actions associated with their roles. Both the Test Conductor and Maintenance Personnel are able to view the system’s administration data. Maintenance Personnel are able to launch the STaTE project.

The following use case diagram (Figure 1) gives an overview of the options different users have when interacting with the STaTE system:

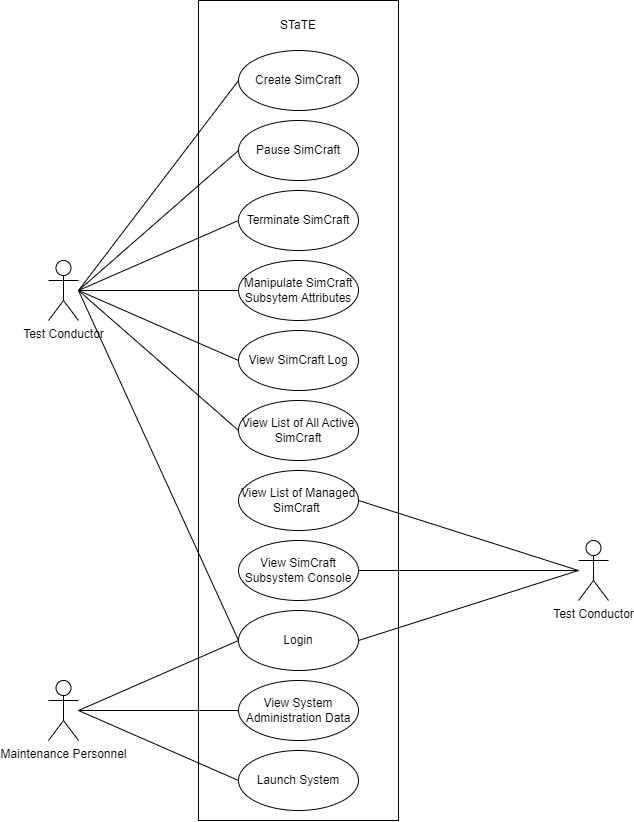


Figure 1: STaTE Use Case Diagram.

### **Design Constraints**

The STaTE project is constrained by limitations related to its operating environment, the django framework, and web security.

Starting with the operating environment, the STaTE project is built to be hosted remotely. This means Developers, Maintenance Personnel, and Test Conductors will not have direct access to the environment the STaTE project operates in. All system functionality has to be accessible remotely through interfaces maintained by the STaTE project. Remote hosting was chosen despite its rigid nature for the benefits associated with resource scalability and system modularity.

The django web framework also provides constraints on the STaTE project. These constraints are related to the speed at which applications built using the django framework operate. These web applications rely on a large number of database operations. Database operations are relatively slow and costly for the system. The django framework was chosen in spite of this because the design of the STaTE project requires many database operations for SimCraft manipulation and django web apps are quicker to implement than many other frameworks.

The last constraint is building the STaTE project around internet security. Being accessible through the web, the STaTE project is made vulnerable to attack from anyone with internet access. Robust protection of user and administrator data is required. Also, actions that are intended to be accessible only to specific users and user-groups must be locked behind authentication barriers. A web-application was chosen for the STaTE project despite these security concerns for its ability to host many users remotely and asynchronously.

### **Future Contingencies**

#### Missions

Contingencies regarding simulated missions may arise in functions of a simulated spacecraft. Differences in spacecraft systems and missions make it difficult to account for all possible mission types.

Some differences in missions may include (but are not limited to):

* Propulsion mechanisms
* Mission pathing
  + Orbits vs landings
  + Orbits around Earth vs orbits around other bodies vs orbits around the galaxy
* Energy systems

#### Browser Compatibility

Contingencies regarding browser compatibility may come up due to large updates in modern web browsers. Currently, the website is being developed to be used on the four most popular browsers that comprise over 90% of all desktop users. Possible incompatibilities may occur based purely on what libraries a browser supports. An example of a common issue that arises because of outdated browsers comes from the CSS nth child programming, where Internet Explorer 12 and Edge both support the function, but Internet Explorer 11 and all other previous versions do not have support.

## Document Organization

This document is laid out to give developers, maintenance personnel, and administrators an overview of the STaTE project. Section 1: Introduction gives a high level view of the STaTE system and the SDD itself. Section 2: System Architecture describes the general architecture of the STaTE project and the detailed architectures of STaTE’s subsystems. Section 3: Human Machine Interface describes the interfaces human users of the system are provided and how these interfaces interact with the system. Section 4: Detailed Design describes the design of the system in depth and how internal systems communicate with one-another. Section 5: External Interfaces describes how existing systems outside the scope of the project are integrated and how they interact with the STaTE system. Section 6 System Integrity Controls describes the measures taken to protect restricted data and actions within the system.

## Project References

This section provides a bibliography of resources that the STaTE project references.

#### Reference for Console Information:

* How the Mission is Controlled: Inside NASA and Boeing Joint Operations
  + A document containing the type of information NASA tracks during a mission, as well as picture reference for how a mission control GUI looks
  + <https://www.nasa.gov/feature/how-the-mission-is-controlled-inside-nasa-and-boeing-joint-operations>
* Xplore’s Major Tom software delivers satellite operations testing for NOAA with Microsoft Azure Orbital
  + <https://medium.com/kubos-tech/major-tom-mission-ops-for-the-21st-century-329905913911>
  + <https://www.prweb.com/releases/xplores_major_tom_software_delivers_satellite_operations_testing_for_noaa_with_microsoft_azure_orbital/prweb18732185.htm>
  + <https://www.xplore.com/services/operations-as-a-service/major-tom.html>

## Glossary

**ERAU** - Embry Riddle Aeronautical University. The University at which the STaTE project was developed.

**Flight Director (FD)** - Role in the program in charge of the coordination of a Flight Operation team. The user this role is intended to be filled by either students or a professor.

**Flight Operator (FO)** - Role in the program in charge of a specific subsystem aboard a SimCraft. The user this role is intended to be filled by students.

**Simulation Engine -** A simulation creation engine to manage multiple SimCraft instances. The Simulation Engine is the interface the Test Conductor interacts with to manage and deploy a SimCraft system.

**SimCraft -** An instance of a simulated spacecraft. A SimCraft maintains attributes relevant to spacecraft flight operators and updates them periodically in accordance with its defined operations characteristics.

**STaTE** - Spacecraft-Control-Center Training and Testing Environment. A web application platform that facilitates a Test Conductor that creates SimCraft to be managed by Flight Operators.

**SWA** - State Web Application. A hosted web application that serves as the platform for the project’s sub-applications and features

**Test Conductor (TC)** -Role in the program in charge of coordinating and setting up for a mission, as well as setting up anomalies. The intended user of this role is a professor.

## System Architecture

This section describes the system and subsystems architecture for the STaTE system.

## System Hardware Architecture

The STaTE system consists of a few connected devices. Some of these devices are vital for the system to work properly, while others are interchangeable and will be explained below. Due to the limited nature of hardware for this project, section 4.1 depicts a brief diagram of this architecture.

**User Components**

The expected hardware components that the users (Student and Professor) will have to provide is a way to connect to the website, either a desktop/laptop computer or mobile devices (such as a cellular device or a tablet), with a way to input keyboard commands. If the student/flight operator device fails, the system will continue to function for all other users. If the admin/test conductor device fails, the system will continue to function for all other users. However, the adding of anomalies functionality will fail until the admin is able to use a device that is not failing. The user must use a device that can run a browser from the following list:

* Safari
* Google Chrome
* Firefox
* Microsoft Edge

**Developer Components**

For the system to run, a cloud server hosted by Microsoft Azure is utilized. In the event the server fails, the website application will experience temporary downtime. The hardware infrastructure of the server is provided by Microsoft, therefore this system relies on the ability of Microsoft to maintain availability of the resources.

## System Software Architecture

The software is currently organized into two sections: the web application - SWA and simulation.

**Web Application**

The Web Application as structured by Django is the general structure that allows the SWA to function as the web application.

**2.2.1.1 SWA**

The SWA side of the software is written in Django.

**a.) Home**

**b.) FO**

**c.) TC**

**2.2.1.2 Simulation**

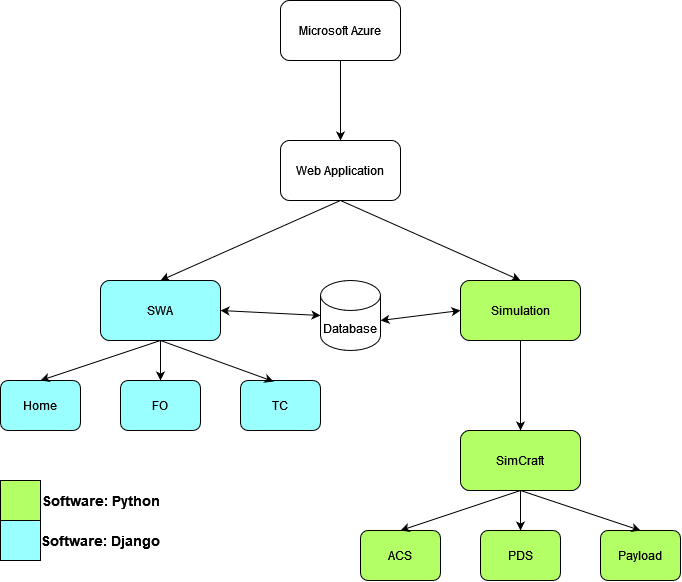
The Simulation side of the software is written in python.

**2.2.1.2.1 Simcraft**

**a.) ACS**

**b.) PDS**

**c.) Payload**

****

**Software Structure Diagram**

The above diagram shows how our software is organized based on the languages we are choosing. Django is a high-level Python web framework, where some components are created in python and others are Django. The blue boxes above are written in Django and the green boxes are written in python. This diagram does not show data flow, but just a high-level hierarchy.

## Internal Communications Architecture

The internal communications of this system is going to be handled mostly by the database. The Test conductor will input simulation data when creating a simulation. When the simulation is started by the test conductor the stop\_flag will be set to 0, and the simCraft will gather simulation data via the database. The simulation will check the database for Test Conductor changes periodically during the simulation run time. The simulation can be stopped when the stop\_flag becomes 1. The Test Conductor can change the stop\_flag by choosing to stop the simulation, which will write to the database and will be found when the simCraft checks for database changes. Thus, the Test Conductor will never directly send data to the simulation or simCraft. Alternatively, the SimCraft will never directly send data to the Test Conductor, avoiding it via sending reports and data to the database.

## Human-Machine Interface

This section provides details of the inputs and outputs as they relate to users operating the web application.

## Inputs

There are three user roles that provide inputs into the system: Flight Director, Flight Operator, and Test Conductor.

Flight Director

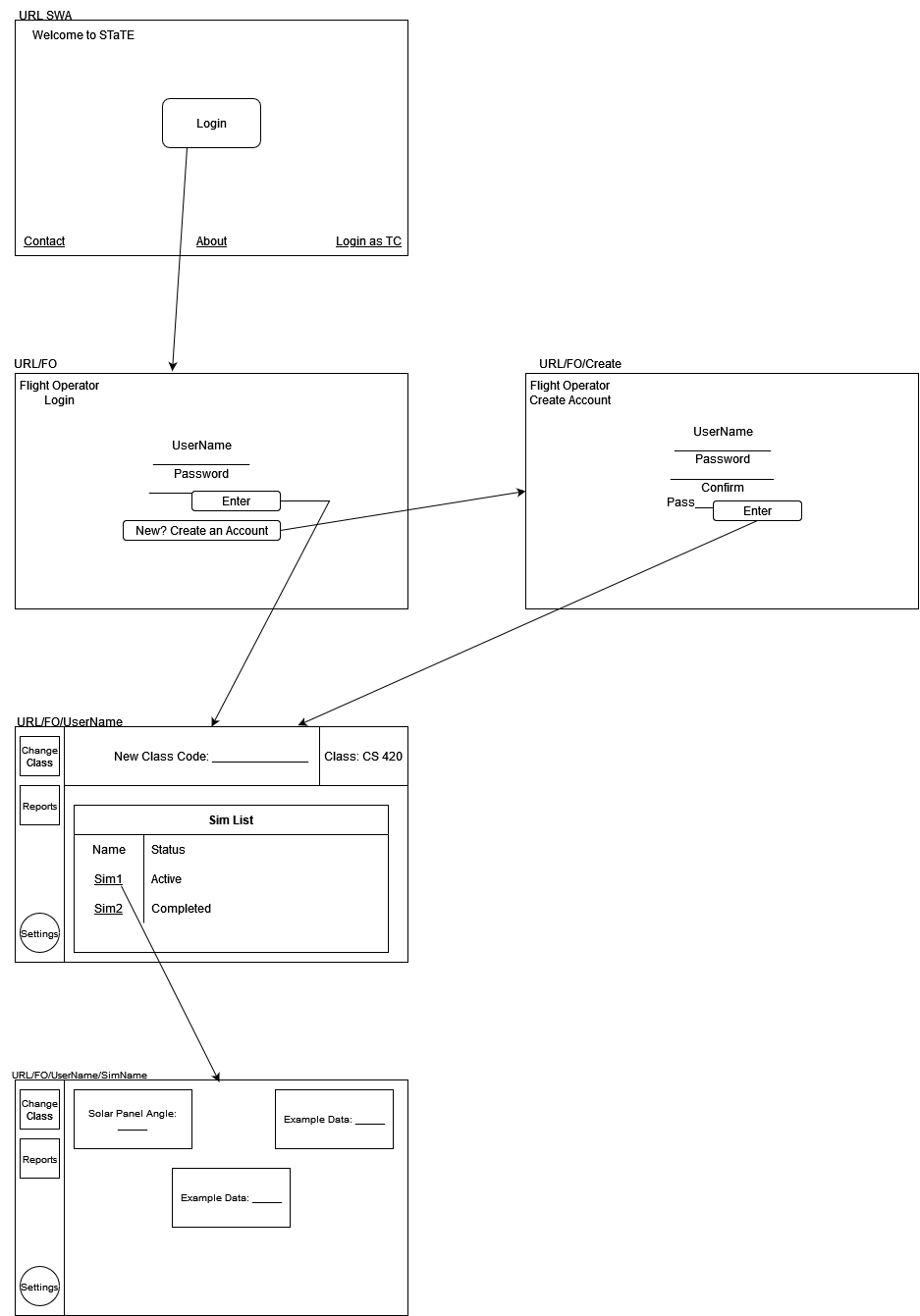
* For the purpose of development, the flight director role is temporarily ignored and will be resumed later in the project.

Flight Operator

* Issue out commands for adjusting components of their assigned subsystem of the spacecraft
* Enter login credentials
* Clicking of many button options for navigation
* Enter class code
* Change class

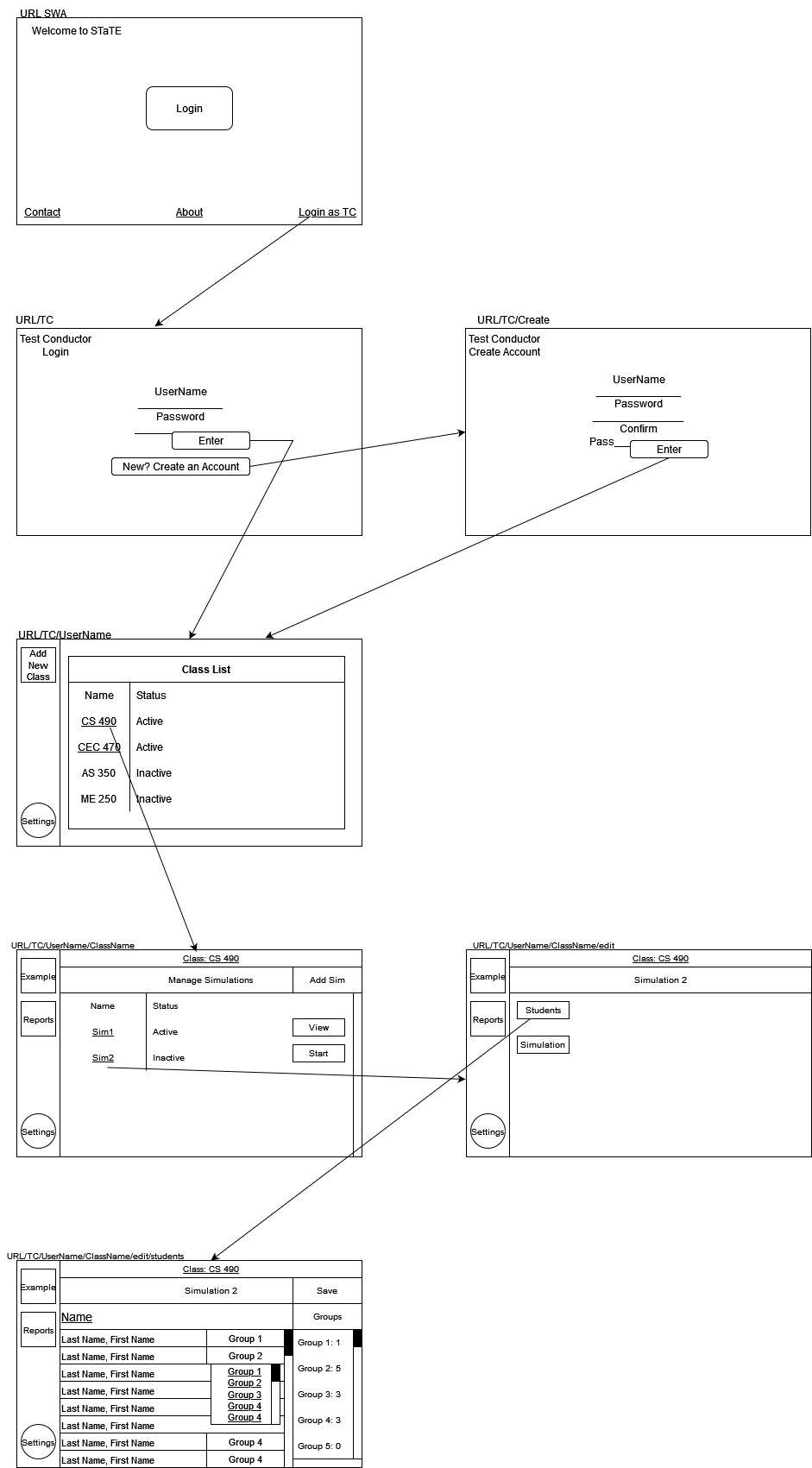
Test Conductor

* Issue out commands for subsystem anomalies in the spacecraft mission
* Issue out commands to create new types of spacecrafts
* Assigns roles/subsystems to the Flight Operators
* Change class
* Create new class
* Enter login credentials
* Clicking of many button options for navigation

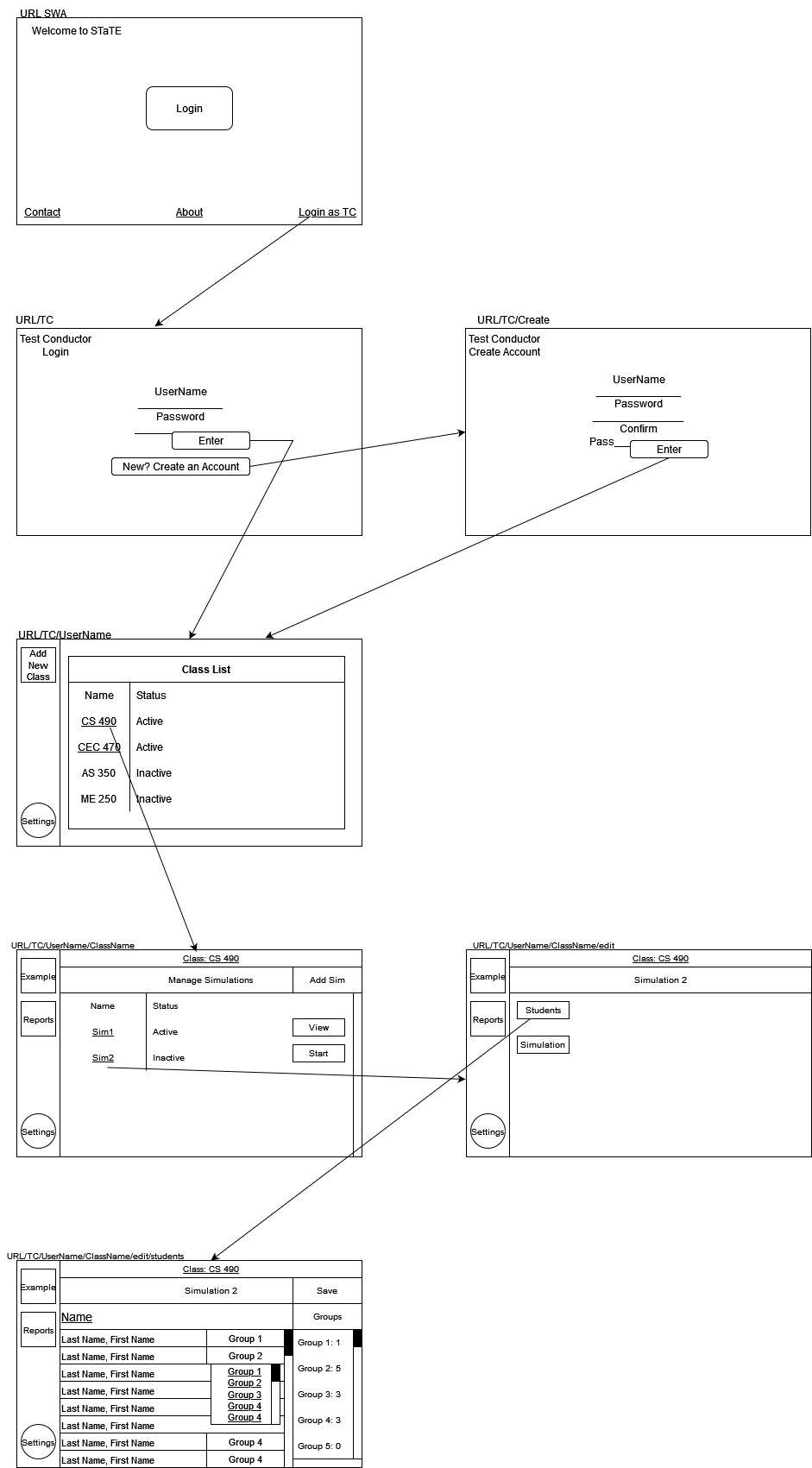


**FO User Interface**

The above diagrams show the basic GUI design for the SWA and the Flight Operator after login has been achieved. As can be seen, the pages communicate via many buttons on the screens. The arrows point to the screen that will be brought up when that button is clicked. User input can be interpreted as button clicks and text box entry. The output of these types of input are usually a page change, except in the case of the simulation. There are interfaces that are still being decided and will be depicted in later versions. The vertical dashboard shown in the images above is also theoretical.



**TC User Interface part 1**



**TC User Interface part 2**

The above diagrams show the basic GUI design for the SWA and the Test Conductor after login has been achieved. As can be seen, the pages communicate via many buttons on the screens. The arrows point to the screen that will be brought up when that button is clicked. User input can be interpreted as button clicks and text box entry. The output of these types of input are usually a page change, except in the case of the simulation. There are interfaces that are still being decided and will be depicted in later versions. The vertical dashboard shown in the images above is also theoretical.

## Outputs

The simulation will output data in regard to the real time running of the spacecraft

mission simulation, and the GUI (the website) will translate this information back to the

user.

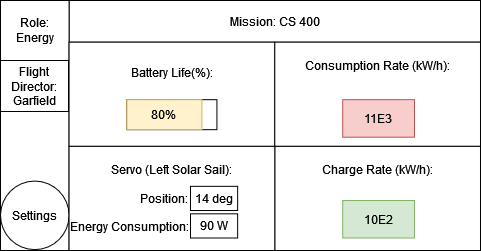


Figure: Example Energy Conductor screen

The GUI relays information to the user regarding high-level system stats (in this case energy consumption and charge rates), and low-level system stats (specific components, i.e. the left solar sail as given in the above example).

## Detailed Design

This section provides the information needed for a system development team to actually build and integrate the hardware components, code and integrate the software modules, and interconnect the hardware and software segments into a functional product. Additionally, this section addresses the detailed procedures for combining separate COTS packages into a single system. Every detailed requirement should map back to the FRD, and the mapping should be presented in an update to the RTM and include the RTM as an appendix to this design document.

## Hardware Detailed Design

The following diagram displays the graphical representation depicting the hardware

items and the relative positioning of the components to each other:

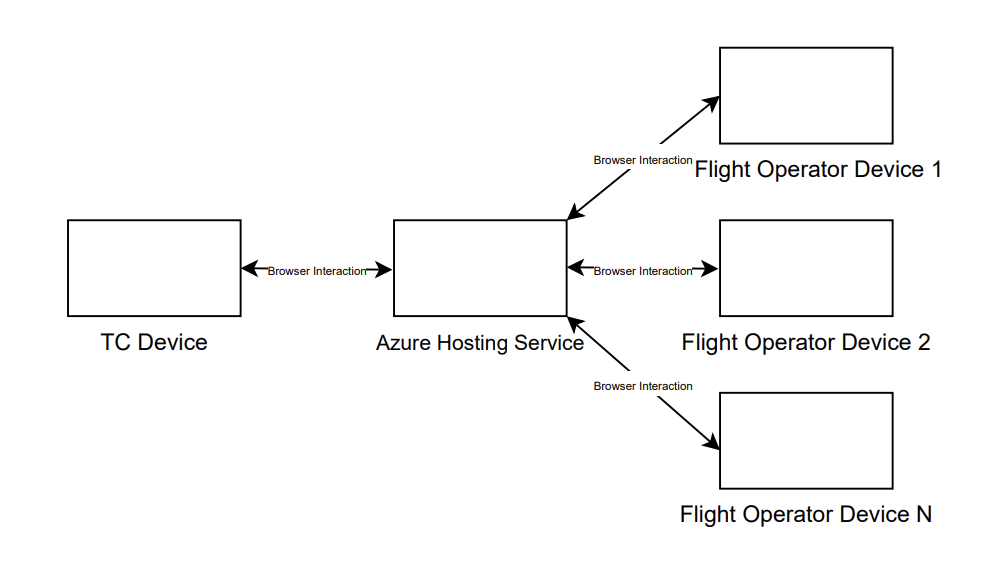


Figure : Hardware Architecture Diagram.

The Test Conductor (TC) Device communicates to the Azure Hosting Service through

browser interaction. From the hosting service, the Flight Operators (devices up to four

per simulation) communicate with the hosting service through browser interaction.

## Software Detailed Design

#### STaTE User Classes

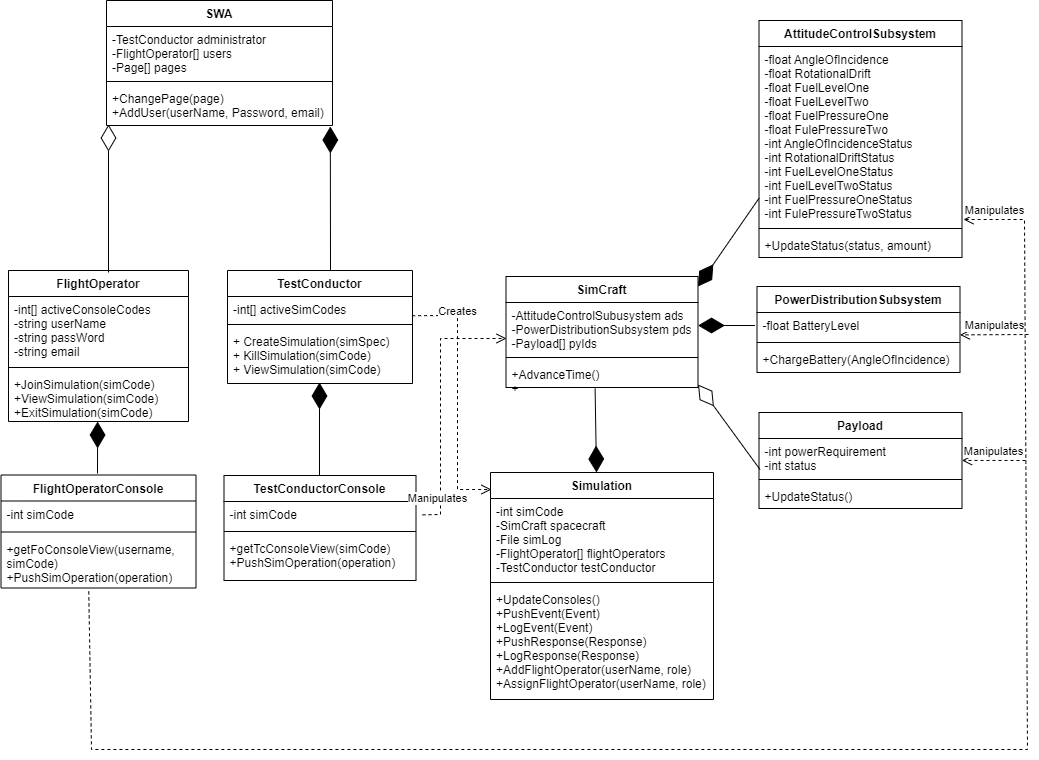


Figure : STaTE UML Diagram.

The above UML diagram shows the layout of the STaTE project from an object-oriented point-of-view. The subsystems of STaTE can be seen as objects interacting with one-another.

**SWA:** The SWA, STaTE Web Application, subsystem is the web application responsible for user management and page navigation for web pages that do not require authentication to access. These pages include a homepage with a brief description of the STaTE project, an about page with a more detailed description of the project, a contact page with information on how to contact site administrators and maintenance personnel, a Flight Operator login page where Flight Operator users can signup or login, and a Test Conductor login page where a Test Conductor user can login. This object maintains one Test Conductor object as its administrator and an arbitrary number of Flight Operator objects as its users.

**FlightOperator:** A FlightOperator object is an instance of a class of users that are able to operate SimCraft that they are assigned to. FlightOperators maintain a list of codes that represent the active simulations they are participating in. FlightOperators also maintain user information such as a username, password, and email. FlightOperators are able to join an active simulation if the simulation’s code is provided, view a simulation given a code for a simulation they are participating in, and exit a simulation given code for a simulation they are participating in. These actions are managed on the FlightOperators homepage which is protected by user authentication. Electing to view a simulation from the FlightOperator’s list of simulations will transfer control to the FlightOperator’s owned FlightOperatorConsole object.

**FlightOperatorConsole:** A FlightOperatorConsole is an object that displays available actions for a FlightOperator user and pushes the effects of these actions to a simulation object in the system’s database. A FlightOperatorConsole displays only the actions available to a FlightOperator for one simulation from its maintained simulation code and FlightOperator username. These actions are displayed graphically based on the characteristics of the simulation and the role assigned to the FlightOperator. The page these actions are displayed on is protected by user authentication.

**TestConductor:** A TestConductor object is an instance of a class of users that are able to create and manipulate SimCraft that are assigned to. A TestConductor maintains a list of codes that represent all of the active simulations. A TestConductor is able to create a simulation given its specifications, terminate an active simulation given its code, and view an active simulation given its code. These actions are managed on the TestConductors homepage which is protected by user authentication. Electing to view a simulation from the list of active simulations will transfer control to the TestConductor’s owned TestConductorConsole object.

**TestConductorConsole:** A TestConductorConsole is an object that displays all available actions for simulation and pushes the effects of these actions to the simulation object in the system’s database. These actions can affect any of the simulation’s subsystems as a TestConductor is given full control over a simulation’s attributes. In addition to displaying a simulation’s actions available to a TestConductor, a TestConductorConsole can fetch and display a log of a simulation’s history maintained by the simulation. This log is fetched through the system’s database.

#### Simulation

The simulation will simulate the following consoles:

* Capsule Communicator
  + Communication system between the crew members and ground control.
  + Not connected to communication between the spacecraft and the command center.
* Command and Data Handling
  + This will be represented when the simulation saves data about the spacecraft and mission.
  + It will be assumed that this system will never fail, i.e. all actions are recorded and all information about the spacecraft and mission is accurate and will always work
* Emergency, Environmental and Consumables Management
  + This will be the response to external anomalies, which will be defined as anomalies that take place outside of the spacecraft and its subsystems.
    - External anomalies may include (but are not limited to):
      * Rogue Asteroids
      * Solar flares and radiation
      * Electrical Pulses
* Flight Data File
  + This is a console that works hand in hand with Command and Data Handling, in terms if the system requires information like the flight path of the mission.
  + All commands are being saved anyway, including navigation and orientation commands.
* Flight Director
  + This is represented by the Flight Director user class
* Guidance, Navigation, and Control
  + This will be the simulated console in control of spacecraft orientation and navigation.
  + Possible simulated anomalies/problems include (but are not limited to):
    - Deviating from a mission path
    - Needing to orient the spacecraft to create propulsion on a mission path
    - Needing to orient the spacecraft towards the sun or another solar energy source to charge solar panels
* Mechanical and Power Officer
  + This will be a simulated console in control of spacecraft power and mechanical subsystems.
  + Possible simulated anomalies/problems include (but are not limited to):
    - Running out of reserve energy
    - A servo that needs to resync calibration
    - Needing to charge battery reserves
* Propulsions
  + This will be a simulated console in control of any propulsion systems on the spacecraft.
  + Some propulsion systems will be more necessary than others depending on the mission and will be programmed on a ‘for mission’ basis
* Recovery
  + This will be a simulated console in charge of recovering from an anomaly.
  + While some anomalies may be to prevent an issue from occurring, other anomalies will be to recover from an anomaly.
* Timeline
  + This will be included in the Flight Data File console, where the time a command was inputted or the position of a spacecraft was recorded will also be recorded.

#### Website Application

There are two primary functions of the website application:

* Direct interaction with the simulation
  + Sending commands
  + Reading returned data
* Verification of identity
  + Registering users
  + Hiding missions not available to specific users
  + Restricting commands based on a user’s role

Verifying and storing a user’s identity allows for commands not available to all users to be segregated based on a role that they hold for different missions. This allows for users to be assigned roles in both the website application (restricting who can create and assign missions) and the simulation (restricting who can cause anomalies).

#### High Level System Overview

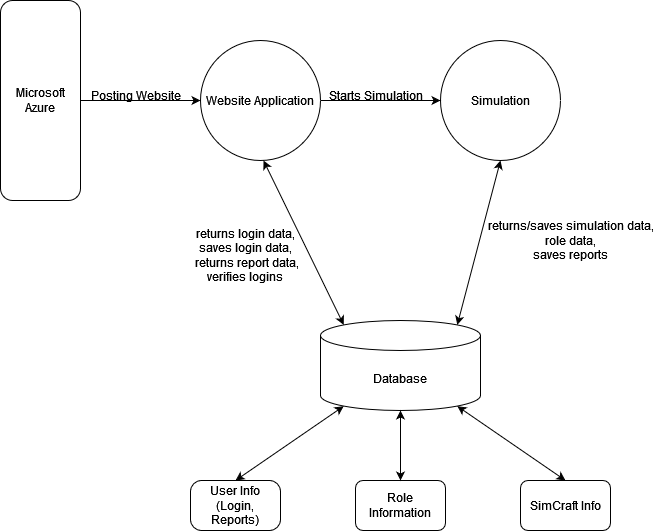


Figure : Data Flow Diagram: Level 0

The data flow diagram above shows the basic way data flows through the system. The Microsoft Azure Server on the left in the diagram hosts the web application. The web application, specifically the Test Conductor can start the simulation. The web application uses the database to save new login data, verify and return login data and get report data. After a simulation is started, the simulation data is gathered from the database. The simulation also saves simulation data and reports.

## Internal Communications Detailed Design

The internal communication system for this project is going to be the server that interacts

with the user and the host. Additionally, the project is going to have to communicate the

simulation with the website, but finding a simulation framework that is written in JS or is

able to be embedded into HTML5/JS will streamline this concern.

## External Interfaces

This section examines any external interfaces that interact with the STaTE.

## Interface Architecture

The following diagram shows the primary means of external communication for the STaTE. Microsoft Azure student portal communicates and manages the web application, where it will call the program software directly from the GitHub repository. Once called, Azure will host and publish the STaTE Web Application within five minutes of building the project. The user will then be able to connect to the program through the URL address. The Web Application and the User Device(s) are two-way communication since both transmit data back and forth.

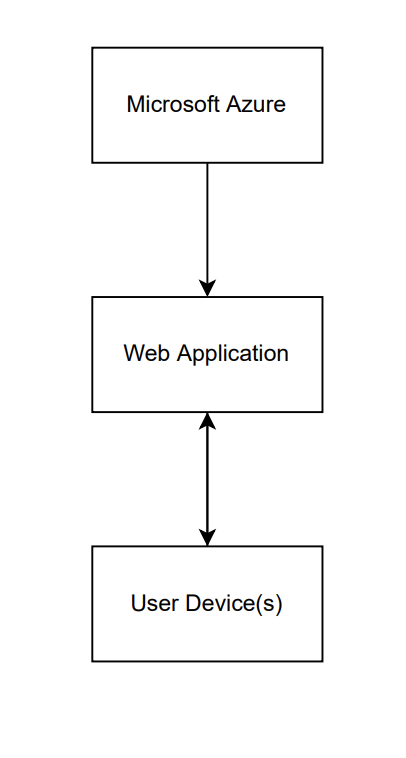


Figure : STaTE Interface Architecture

## Interface Detailed Design

Data will be sent from the simulation to the database using SQLite. The web app will be able to pull data from the database without a need for a change in data formatting. The web app will be able to submit information to the database using HTTP request methods and more SQLite GET/POST requests.

## System Integrity Controls

The only system integrity we can verify for now, without any outlines on any restrictions from IT, is what the team has set on GitHub. Code backups and development versions of the code will be committed and pulled externally from the GitHub for local development debugging. During deployment of the program, the web app will update within five minutes of the edited push, allowing for a clean Continuous Integration and Continuous Delivery (CI/CD) system. The Test Conductor would be the only role to potentially have the ability to access sensitive information from the database, but due to their role in the system, they already would have the ability to access and modify the database, making the security issue irrelevant.