**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 |
| 3.0/Kyle Garber | 12/3/2022 |
| 4.0/Jesse Slager | 3/9/2023 |
| 4.1/Kyle Garber | 3/10/2023 |
| 4.2/Carly Bosma | 3/10/2023 |
| 4.2/Jeremy Mog | 3/10/2023 |
| 4.3/Annamaria Summer | 3/10/2023 |
| 4.4/Jeff Cevallos | 3/10/2023 |
| 4.5/Juliana Altamira | 3/10/2023 |
| 4.6/Jeremy Mog | 4/24/2023 |
| 5.0/Kyle Garber | 4/27/2023 |
| 5.1/Jesse Slager | 4/27/2023 |
| 5.2/Jeremy Mog | 4/27/2023 |

TABLE OF CONTENTS

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

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

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

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

[1.2.2 Design Constraints 5](#_heading=h.scr5bhoz67jx)

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

[*1.2.3.1 Missions 5*](#_heading=h.fhyw55a6lpkv)

[*1.2.3.2 Browser Compatibility 6*](#_heading=h.3j0pj1jgsc7r)

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

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

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

[**2 System Architecture 8**](#_heading=h.s4yg265mf0je)

[**2.1 System Hardware Architecture 8**](#_heading=h.4tpkcqlaq22h)

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

[**2.3 Internal Communications Architecture 10**](#_heading=h.du6ydwmt6wep)

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

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

[**3.2 Outputs 16**](#_heading=h.cy7upx9iqjsk)

[**4 Detailed Design 17**](#_heading=h.hjl88mysx470)

[**4.1 Hardware Detailed Design 17**](#_heading=h.c37sex3nge10)

[**4.2 Software Detailed Design 19**](#_heading=h.p2gxaep9dbt7)

[- STaTE Class Diagram 19](#_heading=h.k4gifocsrcco)

[- Simulation 21](#_heading=h.jgdgy3nekros)

[- Website Application 22](#_heading=h.gxgzn7k8d2nn)

[- High-Level System Overview 23](#_heading=h.43iqxdxqxhbp)

[**4.3 Internal Communications Detailed Design 23**](#_heading=h.sbns2emfgx6d)

[**5 External Interfaces 24**](#_heading=h.8jb3wwy4ie26)

[**5.1 Interface Architecture 24**](#_heading=h.3ph9a08sob0n)

[**5.2 Interface Detailed Design 24**](#_heading=h.oul9p3ugi8ed)

[**6 System Integrity Controls 25**](#_heading=h.oeyuaa6q19zv)

## Introduction

### Purpose and Scope

This document describes the high-level overview, system architecture, operating environment, structure 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 implemented.

#### System Overview

The STaTE project is a platform that allows a Test Conductor to provide simulated spacecraft, referred to as SimCraft, to be managed by teams of Flight Operators. Test Conductors are responsible for first creating classes. Each class is a collection of Flight Operators, SimCraft, and missions. Once a class is created, Flight Operators can join by inputting a Test Conductor-provided class code. Test Conductors create missions by supplying the system with a mission name and target final pitch, roll, and yaw values. After a class contains enough Flight Operators and at least one mission, the Test Conductor can create a SimCraft. To create a SimCraft, a name, mission script, and Flight Operators that will manage each SimCraft subsystem must be provided to the system. Once created, the SimCraft runs until paused or terminated by the Test Conductor. While a SimCraft is running Flight Operators are responsible for completing the activities on their assigned subsystem. Each subsystem has a set of tasks that need to be accomplished to complete SimCraft’s assigned missions. Flight Operators complete these tasks through SimCraft IO terminals that display the allowed actions pertaining to their assigned subsystem. The Test Conductor also can view the attributes of a running SimCraft and download a report of individual Flight Operator’s inputs. The Test Conductor, Flight Operators, and Administrators must log into their respective user accounts to perform the actions associated with their roles. Test Conductor and Administrator users can view the system’s database state. Administrators can also deploy the STaTE application to the internet through a web server.

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, Administrators, and Test Conductors will not have direct access to the environment the STaTE project operates. 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. Web applications using this framework 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 STaTE project utilizes many database operations for SimCraft manipulation. Additionally, Django web apps are easier to maintain and 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 vulnerable to attack from anyone with internet access. Robust protection of user and administrator data is required. Also, actions intended to be accessible only by 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 the 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):

* Attitude changes
* Thermal changes
* Power changes
* Communication/Frequency changes
* Payload objectives

##### 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 web 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.

### **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.

References 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 of this role is intended to be filled by either student or a professor.

**Flight Operator (FO)** - Role in the program in charge of a specific subsystem aboard a SimCraft. The user of 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. The intended use 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 device (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. The user must use a device that can run a browser from the following list:

* Safari
* Google Chrome
* Microsoft Edge

Note: firefox and possibly some other unlisted browsers have proved to have unwanted defects and they should be avoided.

**Developer Components**

For the system to run, a cloud server hosted by Microsoft Azure should be 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 the 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 the Django framework, is the general structure that allows user access to the STaTE system. This includes user account management, data storage, traffic routing, web page serving, and system input/output.

**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.

* SimCraft

a.) Attitude Control Subsystem (ACS)

b.) Electrical Power Subsystem (EPS)

c.) Thermal Control Subsystem (TCS)

d.) Communication Subsystem (Comms)

e.) Payload Subsystem (Payload)

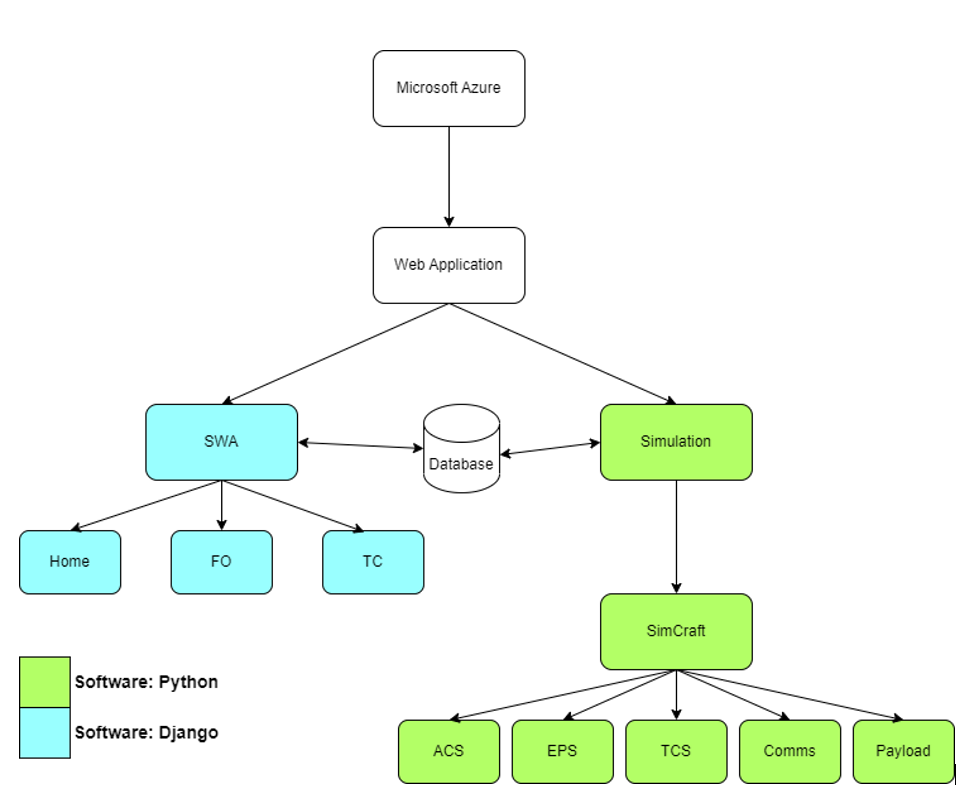
****

Figure 2: Software Structure Diagram

The above diagram shows how the project is organized based on the languages utilized. SWA files exist within the high-level Django web framework, while simulation folder components are created with a standard Python 3.10 format. The blue boxes above are written in Django and the green boxes are written in standard Python. This diagram does not show data flow, only a project structure and hierarchy.

### **Internal Communications Architecture**

The internal communications of the STaTE system are handled through two modalities: database operations and function calls. The Test Conductor inputs required data when creating a simulation. When a simulation is started by the Test Conductor, the stop\_flag is set to 0, and SimCraft will gather simulation data via the database. The simulation will check the database for Test Conductor changes periodically during the simulation’s run time. The simulation is stopped when the stop\_flag becomes 1. The Test Conductor can change the stop\_flag by choosing to stop the simulation, which is written to the database and is found when SimCraft checks for database changes. Thus, the Test Conductor never directly sends data to the simulation, but rather starts and stops it with a function call. Alternatively, 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 STaTE system: Test Conductor, Flight Operator, and Flight Director.

Test Conductor

* Enter login credentials
* Defines subsystem initialization state
* Create a new class
* Create a new mission
* Create a new simulation
* Delete existing simulation
* Assign subsystem roles to Flight Operators
* Change class being viewed
* View active simulations
* Download simulation report

Flight Operator

* Enter login credentials
* Join a class with class code
* Change class being viewed
* View an assigned simulation within a class
* Issues available commands for an assigned subsystem
* View congratulatory message upon simulation completion

Flight Director

* Inherits Flight Operator inputs
* Manages the Payload subsystem

**Home User Interface**

The images below show the GUI design for the SWA home Django app. This app hosts the home page, login page, and registration page. The home and login page is used by both Flight Operator and Test Conductor users. The registration page is used only by Flight Operators signing up as Test Conductor users are created through the Django command line interface.

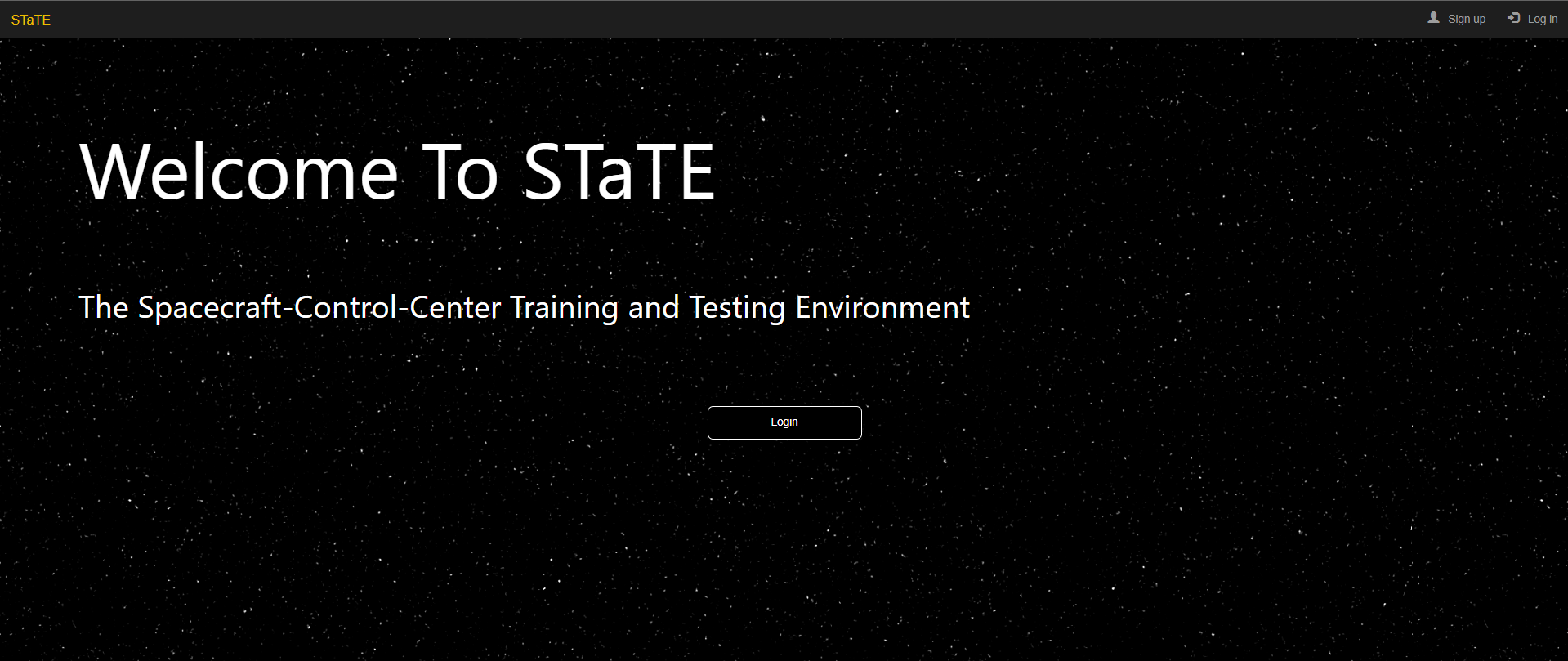


Figure 3: STaTE Homepage

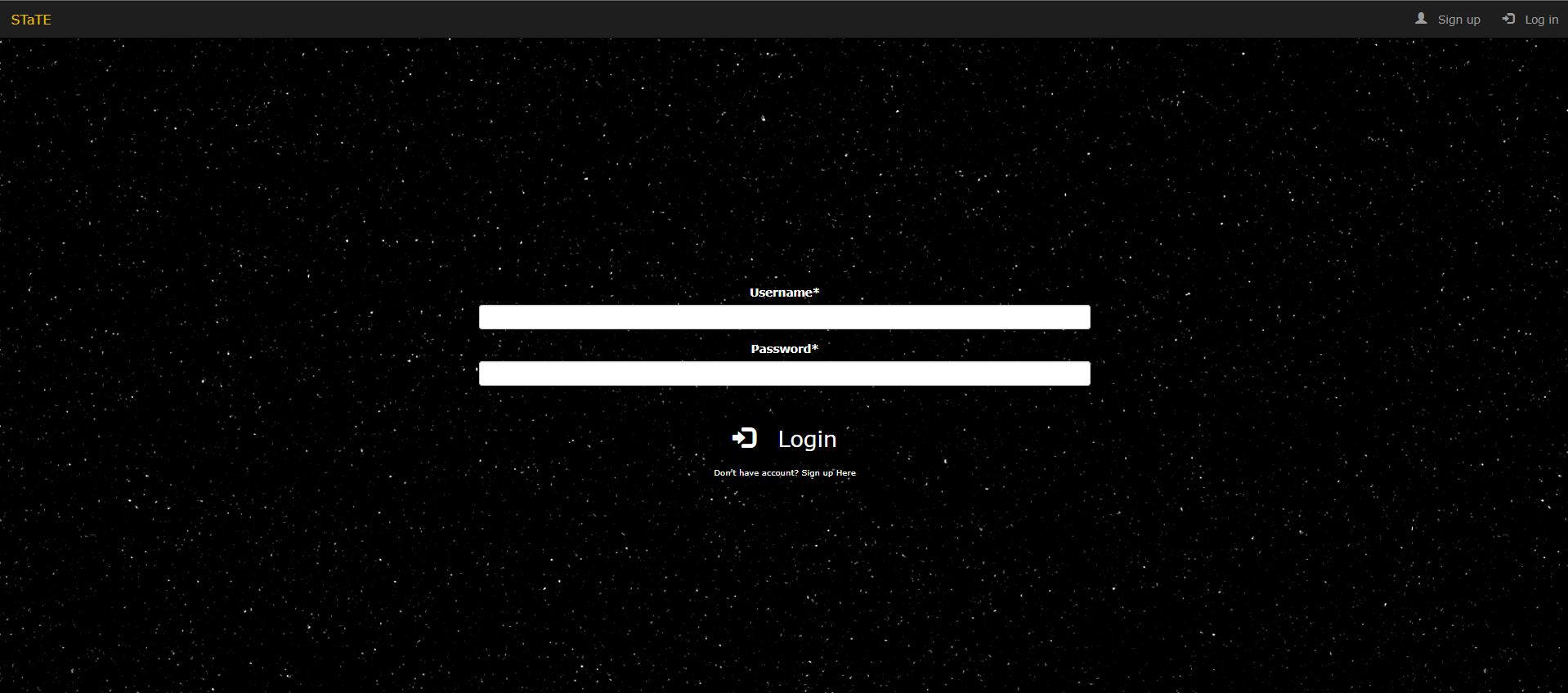


Figure 4: STaTE Login Page

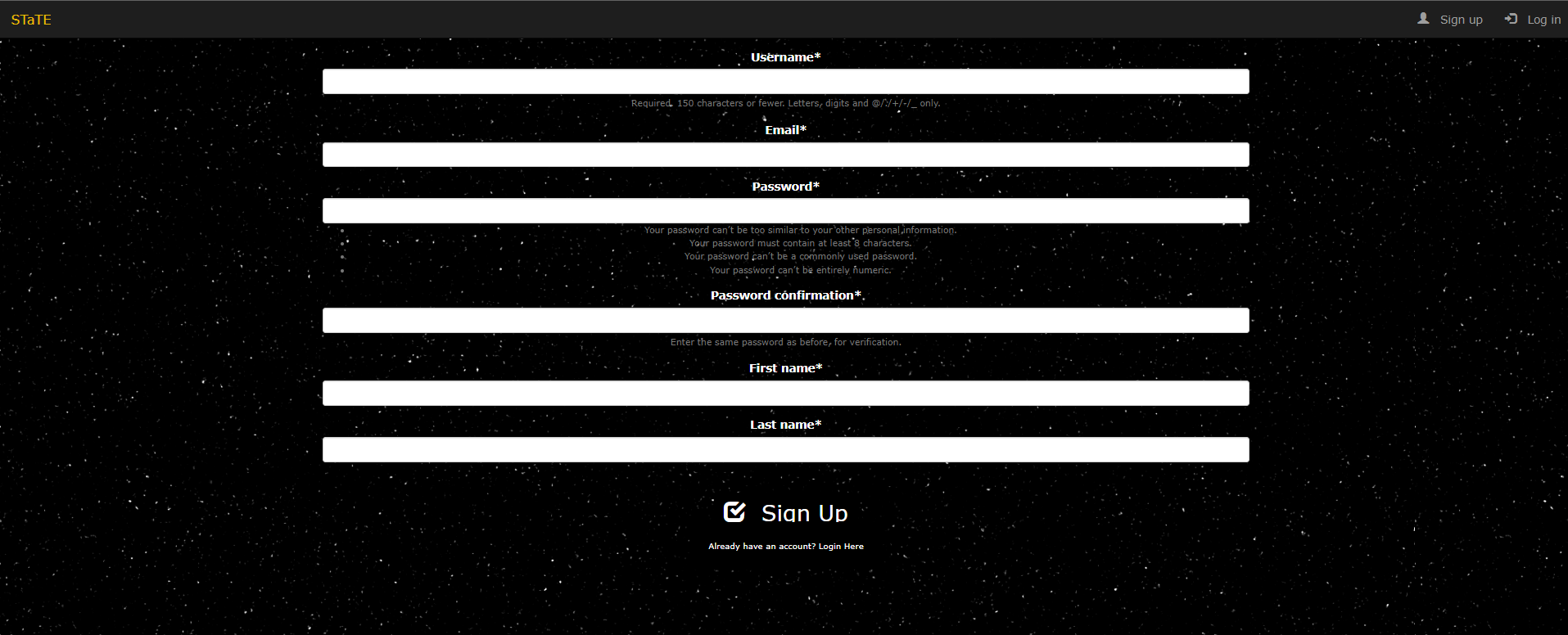
****

Figure 5: STaTE FO registration page

**FO User Interface**

The images below show the GUI design for the Flight Operator user after login has been achieved. As can be seen, the pages are navigable via the buttons on the interface. The arrows point to the screen that will be brought up when that button is clicked. User input can be interpreted as text box entries and command line entries. The output of these types of input is usually page navigation, except when input is to a subsystem terminal. Each subsystem will have its own input terminal along with a synchronized output console that displays the same information to each subsystem. Display containers will also be shown on each subsystem to show real-time updates to each subsystem attribute. There is also a navigation bar on the left-hand side of subsystem dashboards to allow Flight Operators to see but not edit other subsystem attributes within an assigned SimCraft.

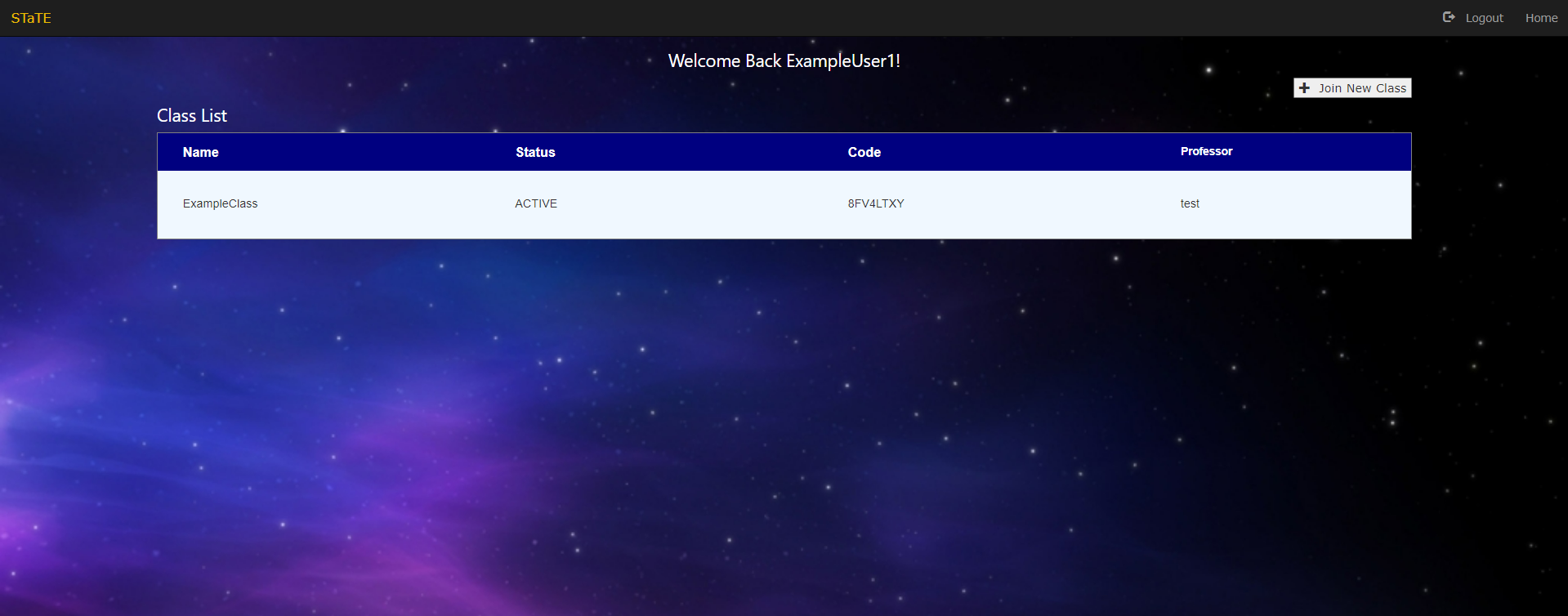


Figure 6: Flight Operator Home Page with one class

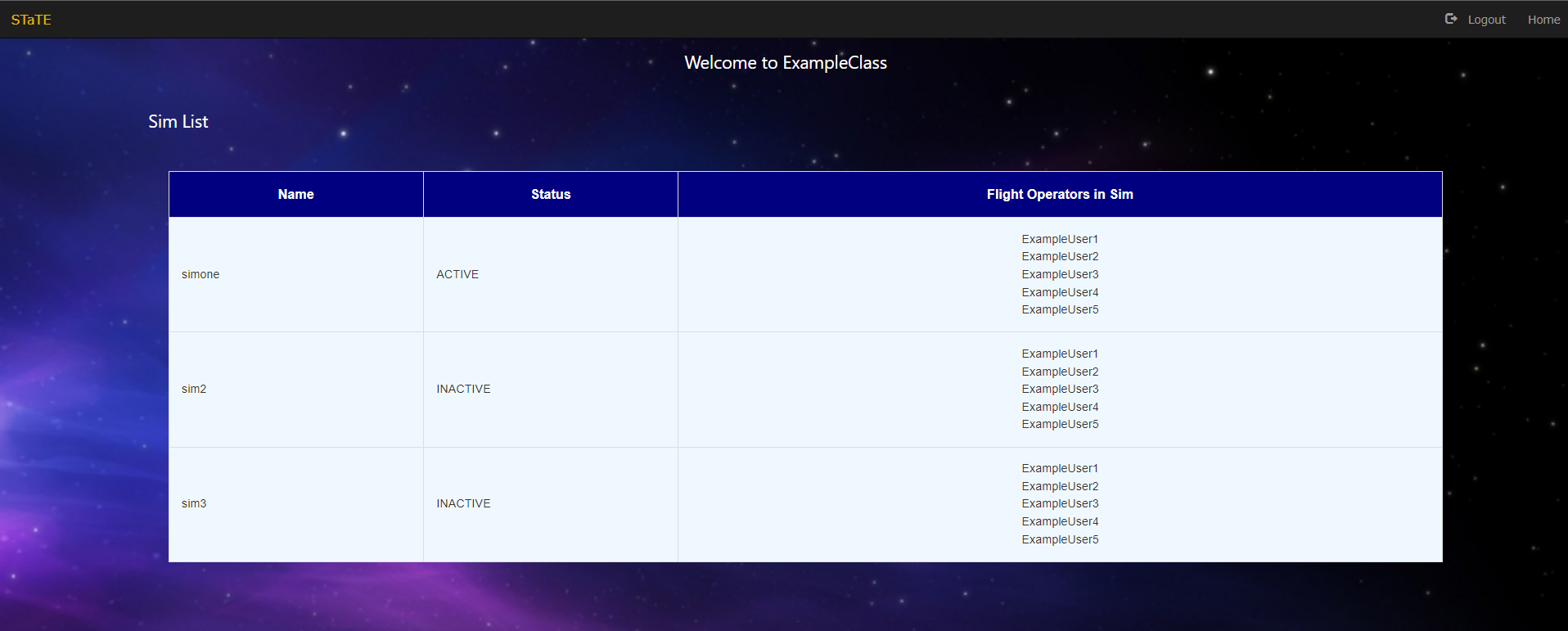


Figure 7: Flight Operator Class Page with three simulations

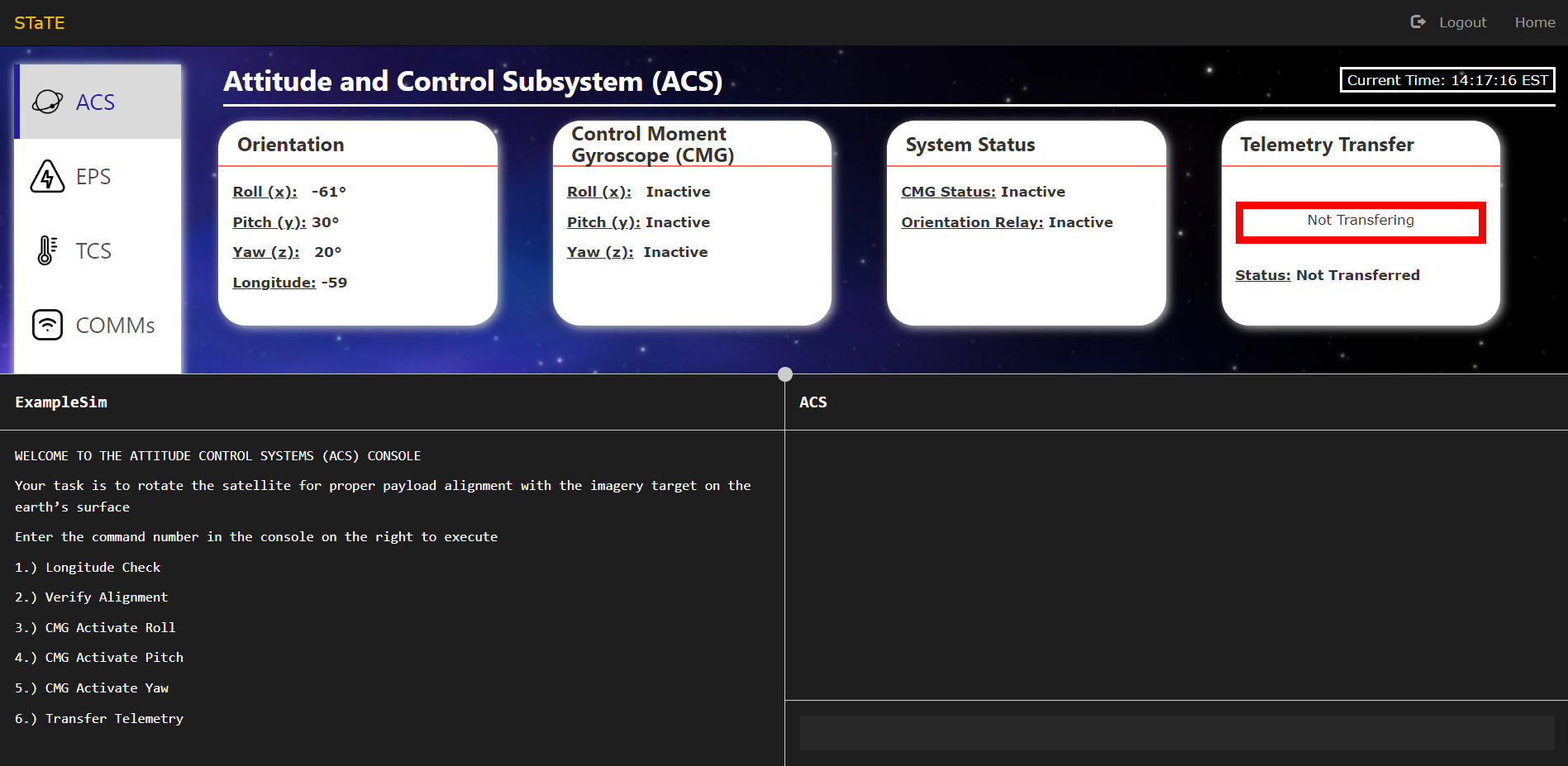


Figure 8: Flight Operator ACS Dashboard

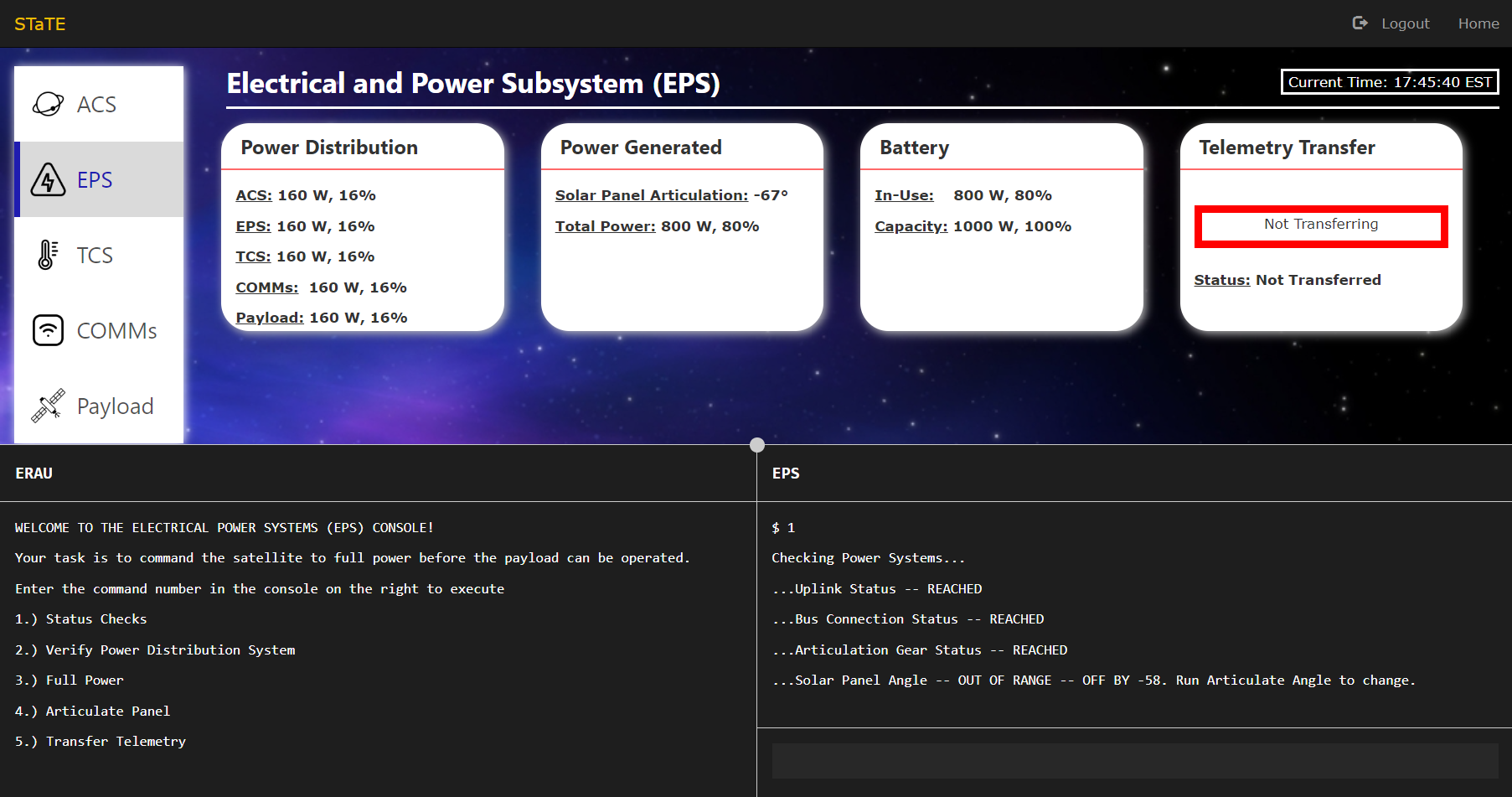


Figure 9: Flight Operator EPS Dashboard

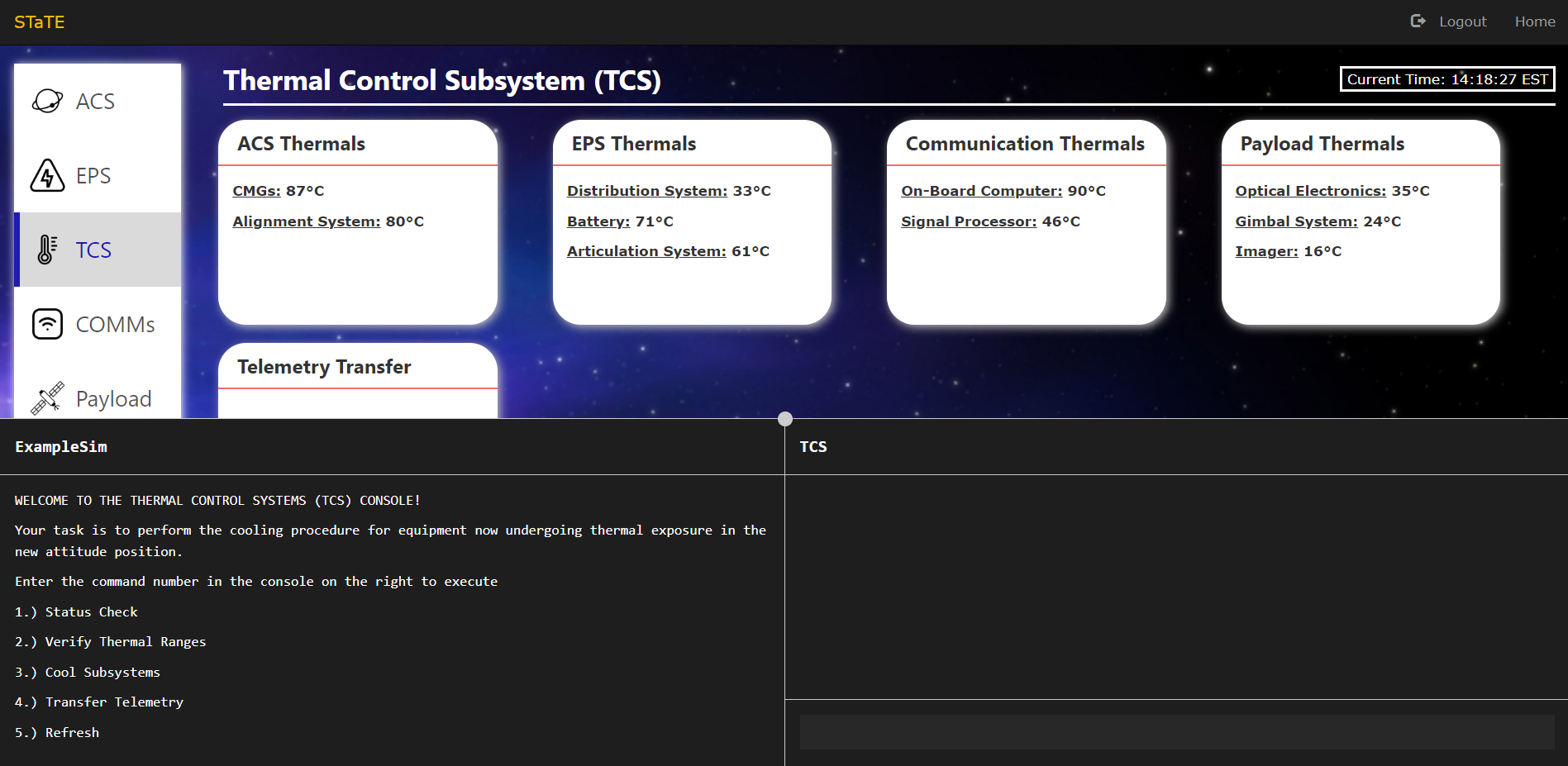


Figure 10: Flight Operator TCS Dashboard

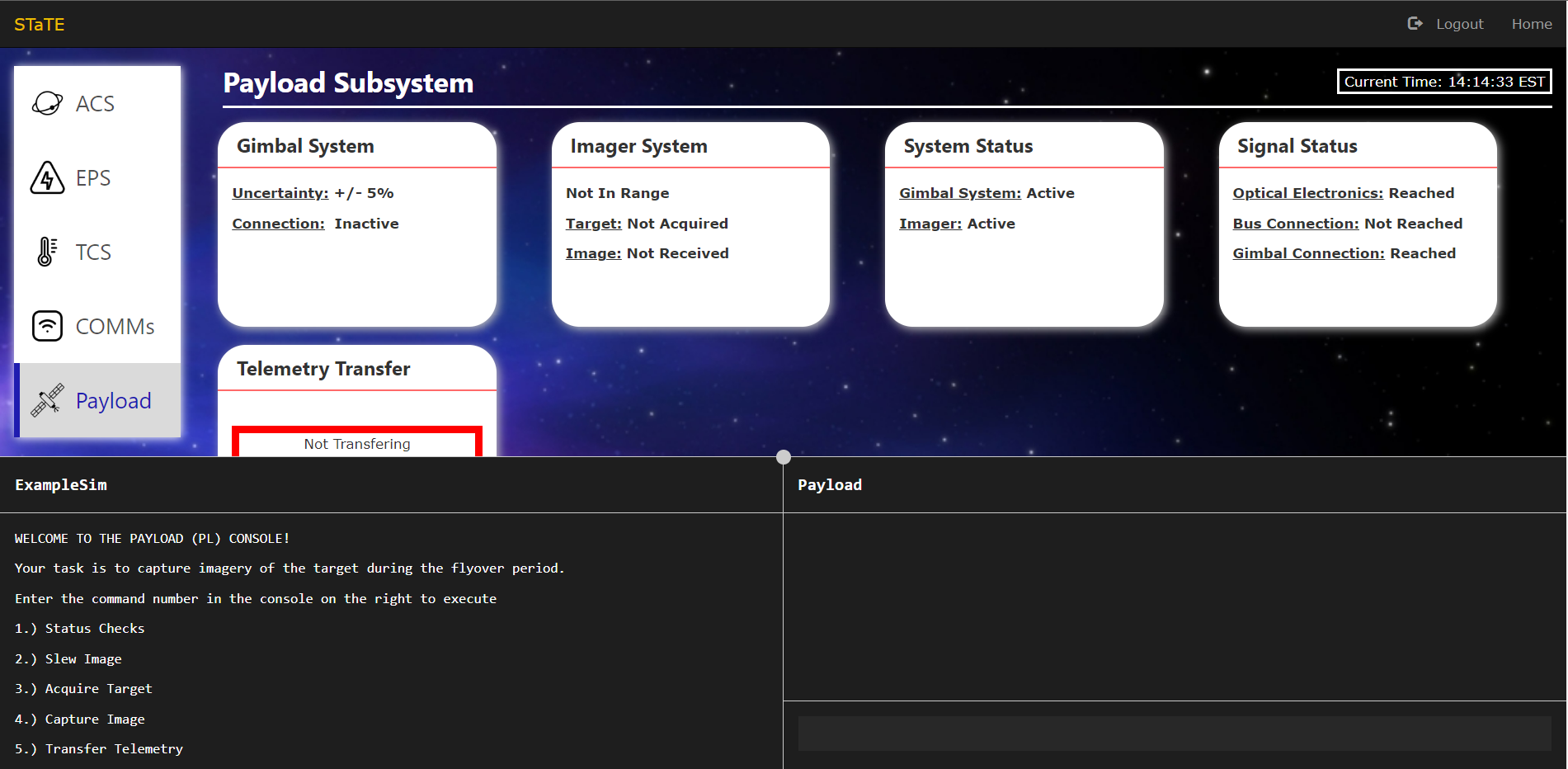


Figure 11: Flight Operator Payload Dashboard

**TC User Interface**

The images below show the basic GUI design for the Test Conductor user after login. As can be seen, the pages are navigable via the buttons on the interface. 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 entries. The output of these types of input is usually a page change, except in the simulation case. The TC will be able to create new classes, missions, and simulations. The TC administration page allows the TC to manage and define classes, missions, and simulations through buttons and textbox fields.

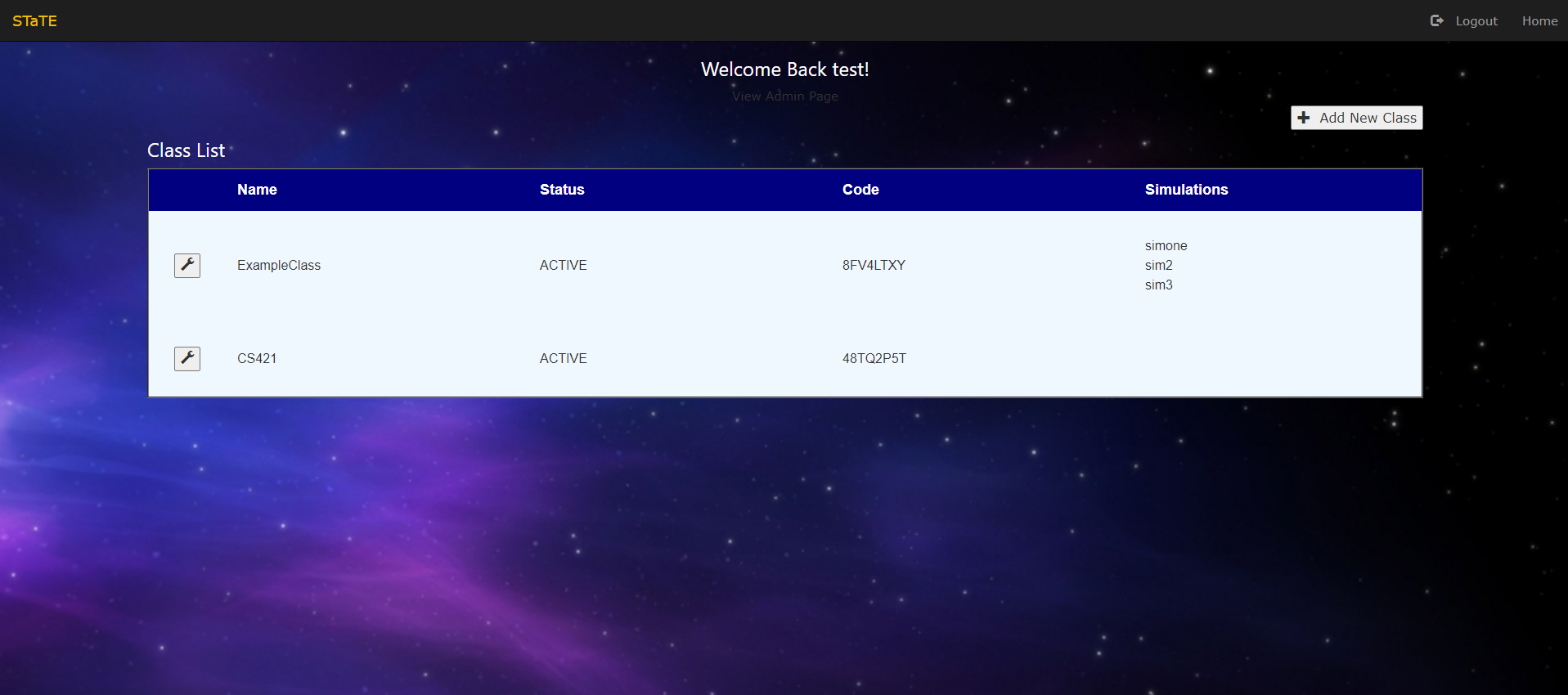


Figure 12: Test Conductor Home Page with two classes

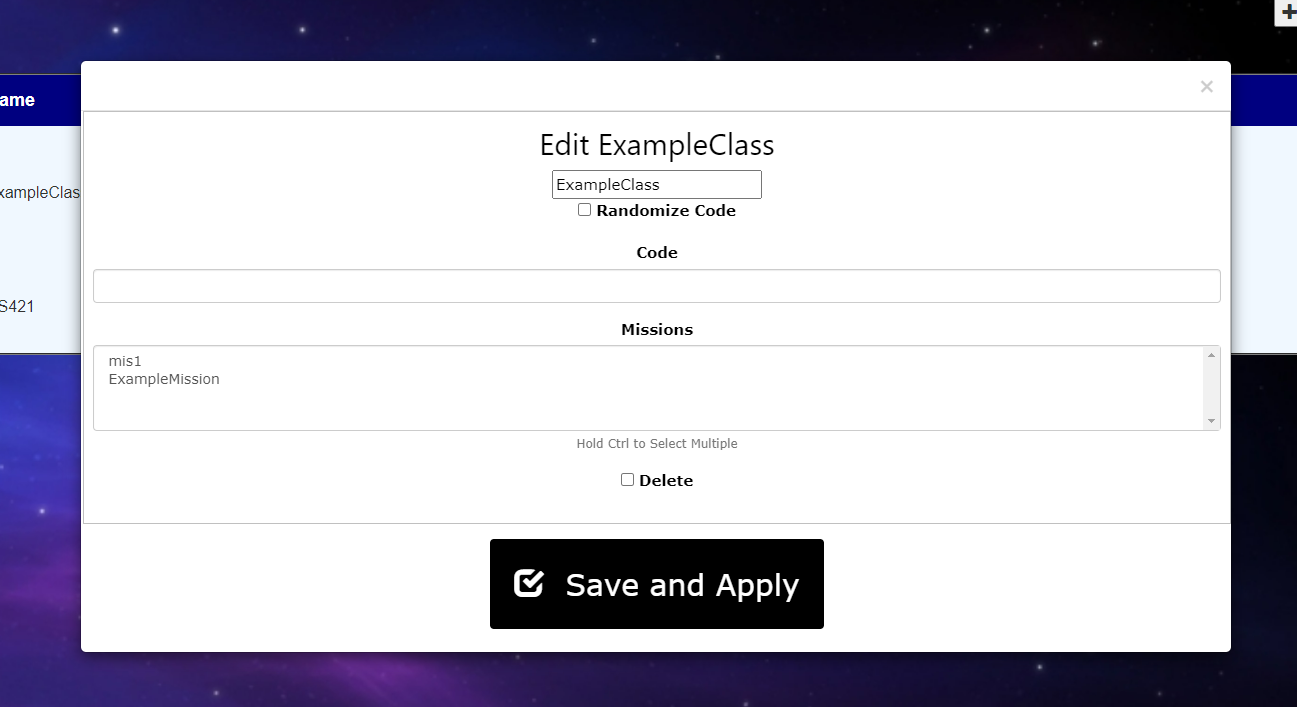


Figure 13: Test Conductor Edit Class Popup Menu

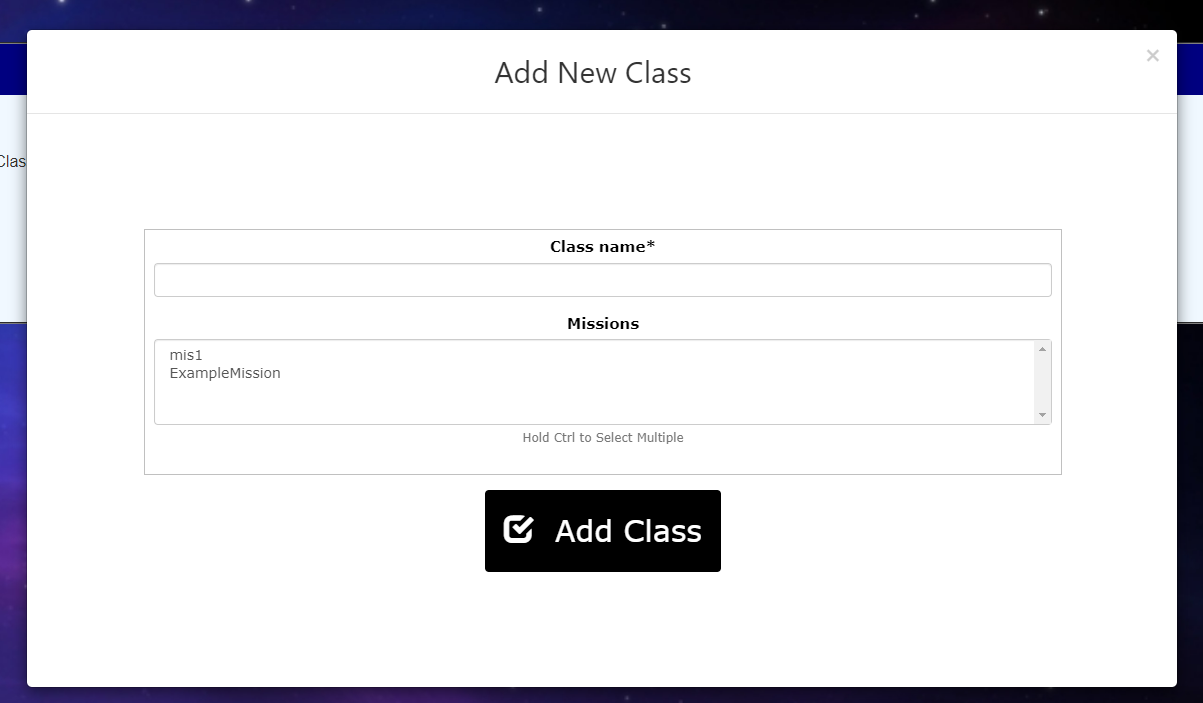


Figure 14: Test Conductor New Class Popup Menu

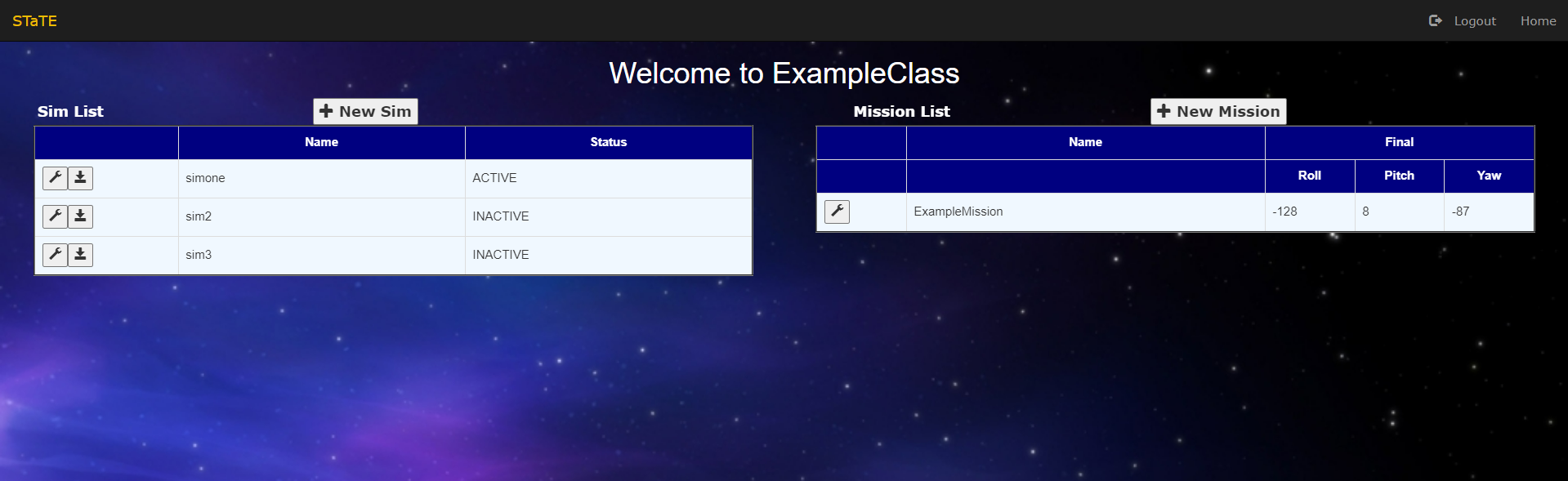


Figure 15: Test Conductor Class Page

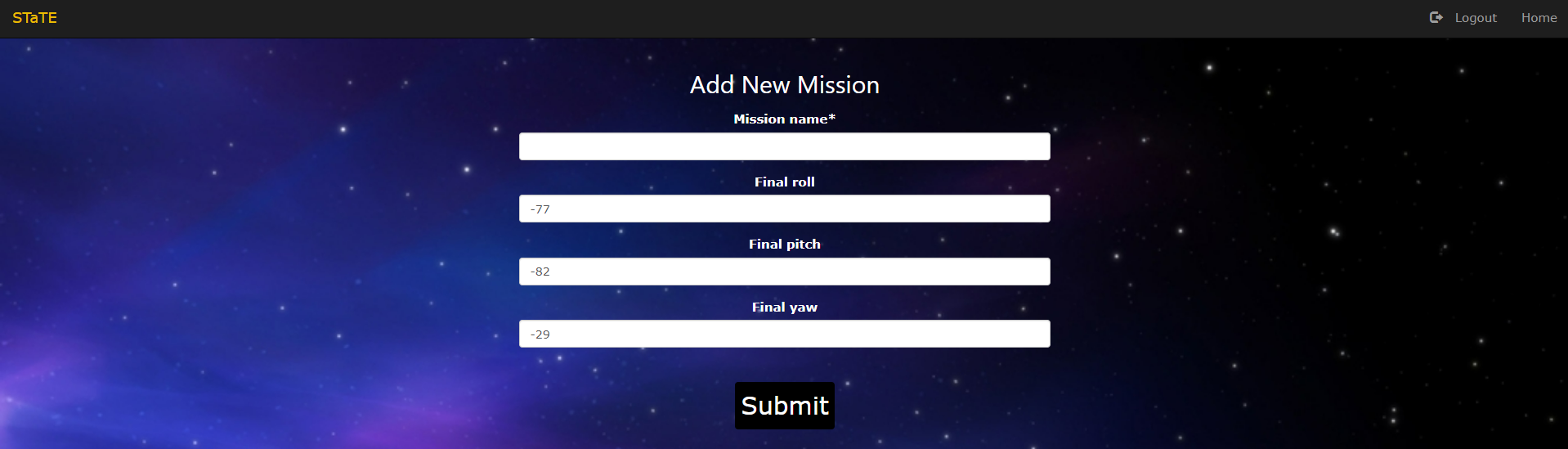


Figure 16: Test Conductor New Mission Page

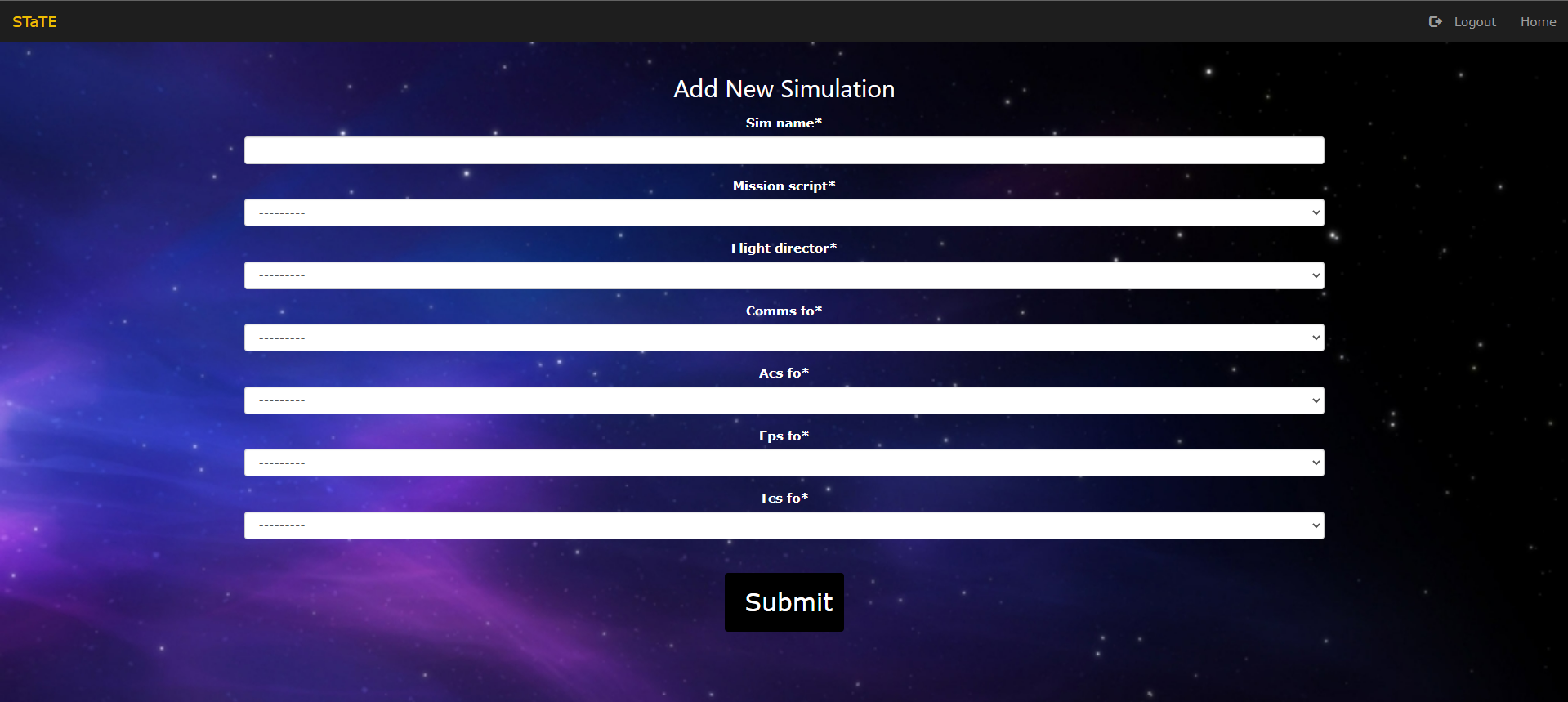


Figure 17: Test Conductor New Simulation Page

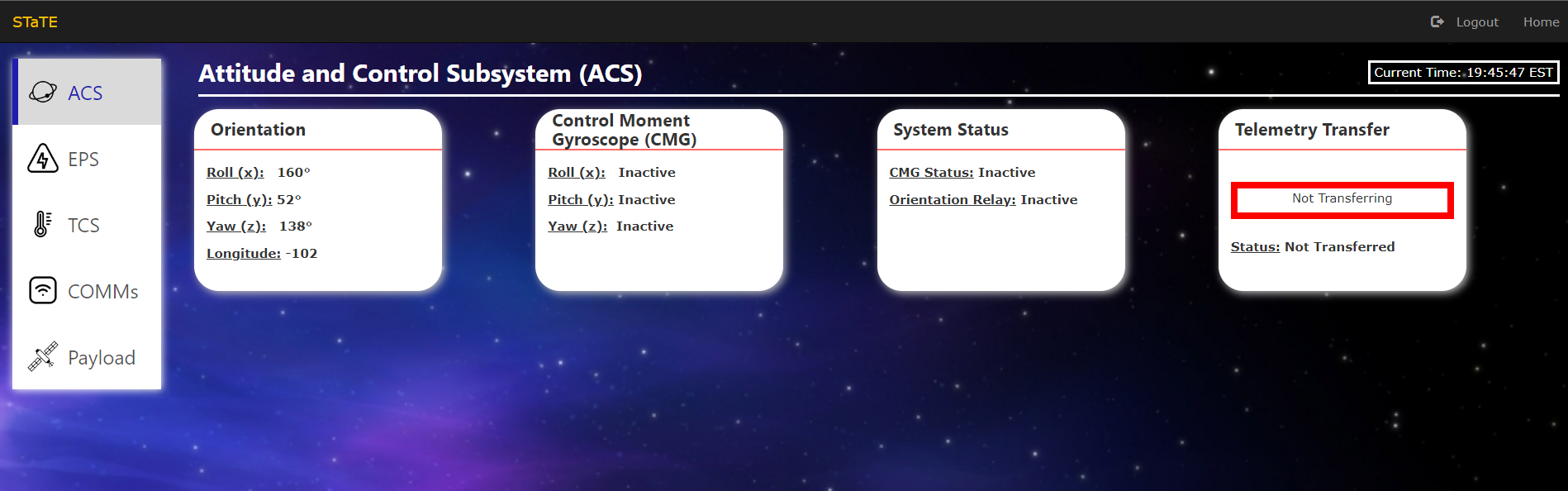


Figure 18: Test Conductor View Simulation Page

### Outputs

The simulation will output data regarding the real-time running of the spacecraft

simulation, and the GUI display this information for the user



Figure 19: Flight Operator Console on Subsystem page

The GUI relays information to the user about the entirety of the SimCraft inside a console output terminal assigned to the Flight Operator along with allowing inputs from the Flight Operator into console script specific to the subsystem they were assigned.

## 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 Commercial Off-The-Shelf (COTS) packages into a single system. Every detailed requirement should map back to the Functional Requirements Document (FRD), and the mapping should be presented in an update to the Requirements Traceability Matrix (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:

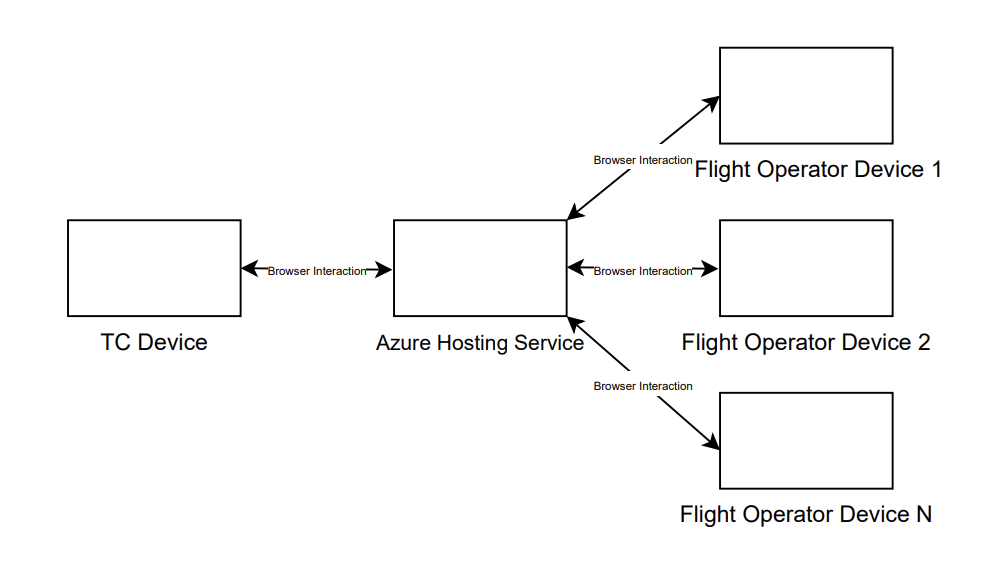


Figure 20: Hardware Architecture Diagram.

The Test Conductor (TC) Device communicates with the hosting service through

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

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

### Software Detailed Design

#### STaTE Class Diagram

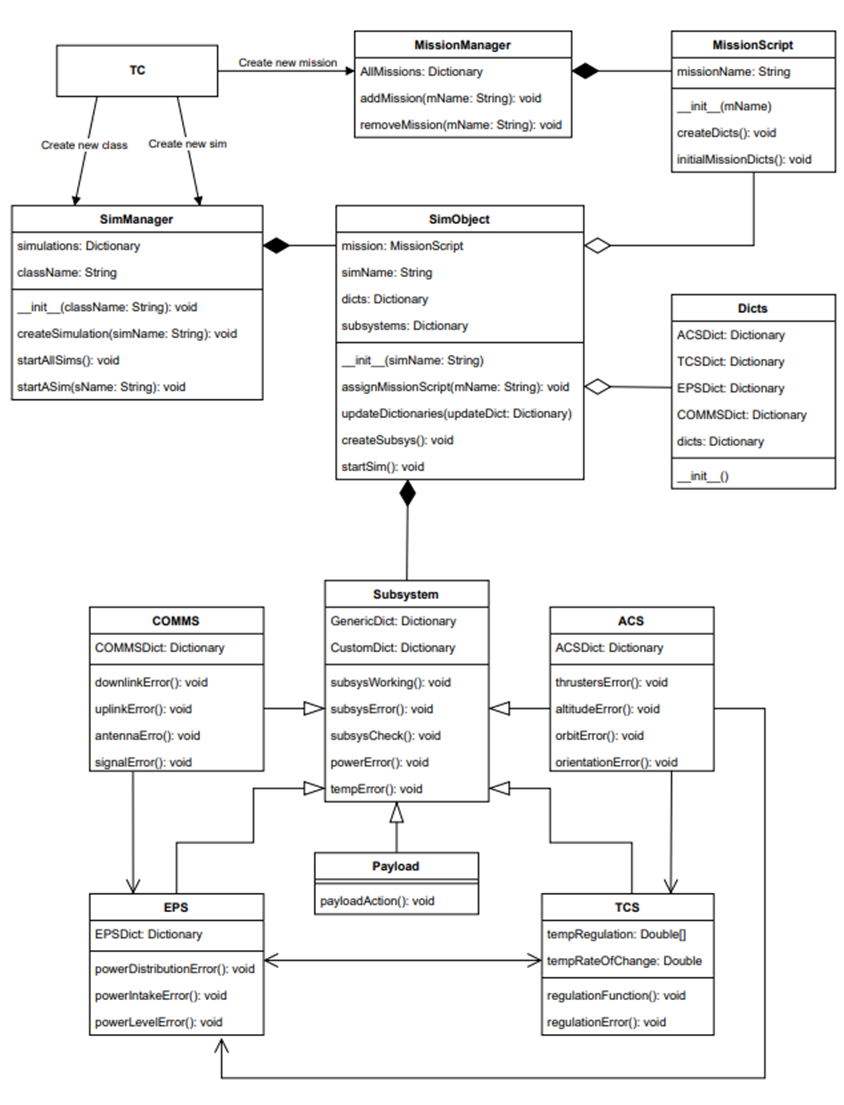




Figure 21: STaTE UML Diagram.

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, and a login page where both Flight Operator and Test Conductor users can log in. The SWA maintains one Test Conductor user with administrator privileges and an arbitrary number of Flight Operator users without administrator privileges. Administrator privileges on the SWA allow a user to access an admin page that maintains a record of all objects within the application’s database

**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.

**FlightOperator:** A Flight Operator object is an instance of a class of users that are able to operate the SimCraft they have been assigned to. Flight Operators maintain a list of codes that represents 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.

**User Consoles:**

**TestConductorConsole:** A TestConductorConsole displays all available actions for simulation and pushes the effects of these actions to the simulation object in the system’s database. The TestConductor can also use this to create new classes, simulations, and missions. 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.

**FlightOperatorConsole:** A FlightOperatorConsole 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 the attributes and values of subsystems to a FlightOperator for one simulation from its maintained simulation code and FlightOperator username. The FlightOperator can manipulate these values using a console showing both their input and output of sent commands. 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.

#### Simulation

The simulation will simulate the following subsystem consoles:

* Attitude and Control Subsystem (ACS)
  + Allows for the rotation of the satellite for image capture
  + Activates the CMGs and perform a 3-degree roll maneuver to capture the imagery target
  + Attributes:
    - CMG Roll, Pitch, and Yaw (θ range)
    - Telemetry Transfer (toggleable)
    - Verify Alignment with the internal computer (toggleable)
* Electrical Power Subsystem (EPS)
  + Checks the system status and commands full power to the satellite in preparation for payload power needs for image capture of a scene on the surface of the earth
  + Although satellites tend to operate at low power when possible, additional power may be necessary for the operation
  + Attributes:
    - Electrical Standby mode for ACS, EPS, TCS, and Payload power systems (toggleable)
    - Solar Panel power level charge and dissipation (toggleable)
    - Telemetry Transfer (toggleable)
* Thermal Control Subsystem (TCS)
  + Performs the cooling and heating procedure for equipment undergoing thermal exposure in a new attitude position
  + Attributes:
    - Cooling functions for:
      * ACS: CMGs and other sensors
      * EPS: Solar Panels and Battery
      * Comms: Transceiver, Antenna, and Amplifier
      * Payload: Camera
    - Telemetry Transfer (toggleable)
* Communications Subsystem (Comms)
  + Verifies a signal lock is established between the Ku-Band satellite antenna and the ground station antenna
  + Transmits the target image to the ground station to process the image and display the results
  + Attributes:
    - Signal (range)
      * Gain
      * Bandwidth
    - Telemetry (toggleable)
      * ACS
      * EPS
      * Comms
      * Payload
* Payload Subsystem (Payload)
  + Based on GPS coordinates, captures imagery of the target during the flyover period
  + Attributes:
    - Gimbel System (toggleable)
    - Photo Capture (stand-alone function)

#### 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.

#### High-Level System Overview

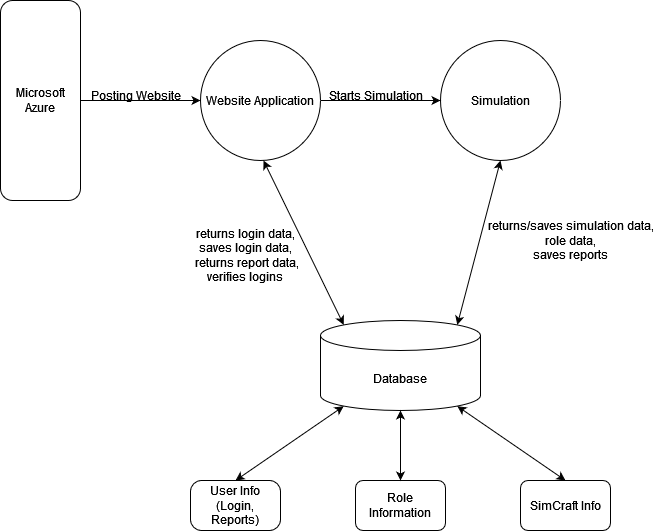


Figure 22: Data Flow Diagram: Level 0

The data flow diagram above shows the basic way data flows through the system. The hosting 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 website and the simulation must communicate, which is achieved via a Django framework paired with an SQLite database.

## 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.

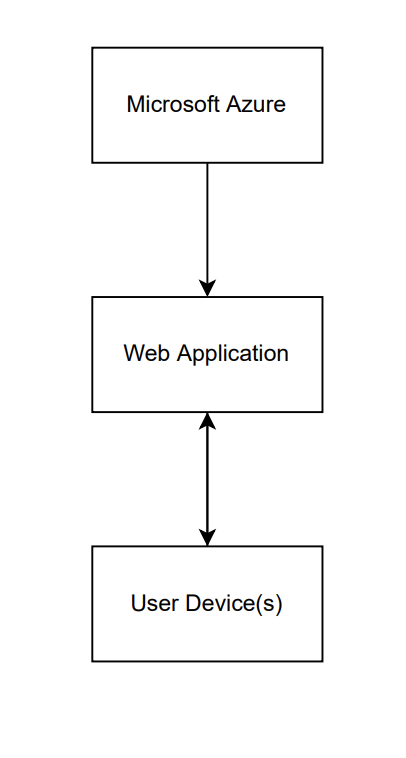


Figure 23: STaTE Interface Architecture

### 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 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.