# 5 Testing

The CyHELICs project focuses on simulation integration between several softwares in real time. Our project proposes a few unique software complications that will need to be tested extensively. Due to the Dockerized environment we will have predominantly unit testing, integration testing, and system testing. Unique challenges that we will face during the testing phase of our design will include making sure that data is properly and efficiently transferred between softwares, making sure that each software correctly interprets that data transferred, and that our system works correctly as an entire system.

## Unit Testing

Unit Testing will be done by making sure that all software packages can return their versions. Checking for errors at this step will make sure that all software packages are installed correctly on their respective Docker containers. Making sure that Docker properly sets up the shared backend “/app” between the Dockers is also critical, but easy to do with checking the Docker shared volume directory. Ensuring that all Docker containers are able to talk to one another is vital, this can be done through a series of pings on localhost ports to ensure that they are opened by the Docker daemon. We can achieve this by a bash or python script whose results can be shown on the front end.

## Integration Testing

We need to keep close track of what information is being sent between the Docker containers in order to make sure that there are no mistranslations in the system. We will set up the expected values to be sent out of the containers and make sure that valuesF are the correct type (integers are not strings, etc.). We can do this as simply as a Python script.

## System Testing

We will test HELICS, as HELICS requires all software (pandapower, python\_dss, Docker, Flask frontend) to be correctly integrated with each other to work. Since HELICS is a bridge software, if one of the software components does not work, HELICS will not function properly. The results of HELICS will be displayed on a Flask frontend application, which will further display the proper connection between HELICS and the frontend.

## Regression Testing

We are using Docker to ensure that new additions and features can be added at any point. It is rather easy to add a new Docker container to the existing nodes that we have set up. Altering the code will be easy to add new features with Docker as its connections will be made over localhost. We will test the localhost ports for the connections being established.

We need to ensure that the existing features are not broken by new Docker containers being added, but due to the nature of the hardcoded port values, unless the new addition is misconfigured, it will run as intended.

## Acceptance Testing

Our client has been working with us throughout the semester to ensure that we are using the correct tools in order to complete the tasks. Building the correct framework out of the correct libraries is critical to getting the job done. We will test whether the requirements are met to the client’s satisfaction and see if the pre planned attacks fulfill the planned goals. This will be done by setting a baseline to achieve success using a small grid. The small grid in question is called “123Bus,” it’s a small scale electric grid model that has 123 bus connections. 123Bus is a small example model provided by OpenDSS.

## Results

Our tests ensure that our product can continue to be developed and help users determine the security of their power grids, regardless of their system specifications or tech stack. For compliance with the project, we must make sure that all parts of the system are able to function individually and communicate with each other - which will lead us to a proper simulation. We can test this through a myriad of ways and present it to the user when a test is successful at affecting the grid. For example, when a certain test is established the diagrams produced by PandaPower, OpenDSS, and Open DER will adjust to those conditions so we will see changes in the model with line thickness (shown on the first figure) and the data results on each individual lines (shown on the second figure).

