ISTANBUL TECHNICAL UNIVERSITY
Faculty of Computer Science and Informatics

SpaceXhibit
Flask Web Application

ITUDB2232 PROJECT REPORT

Hakkı Arda Çoha 150170111
Beyza Aydeniz 150200039
Abdullah Asım Emül 150190016
Alp Türkbayrak 150200108
Ahmet Metehan Yaman 150140030

FALL / 2023

CONTENTS

1 USER GUIDE

SpaceX has many launches and assets, our application makes SpaceX's public API data accessible to the wider public. Users can learn about launches rockets, ships, capsules, cores, payloads, launchpads and their various details. Moreover, logged in users can add new data, delete and update existing ones, and search whatever they want through our SpaceXhibit application.

Users who have not previously registered in our application can create memberships from the "Sign Up" page and manipulate SpaceX data.

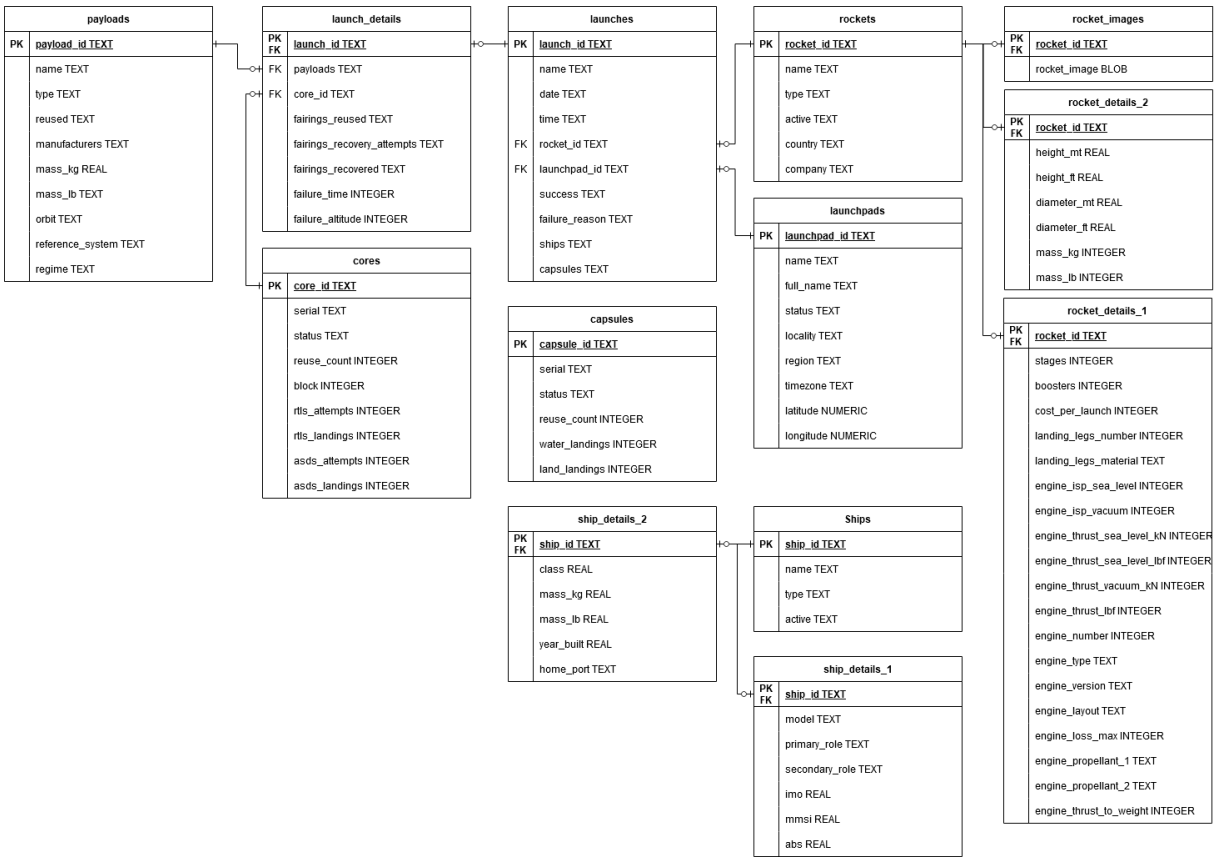
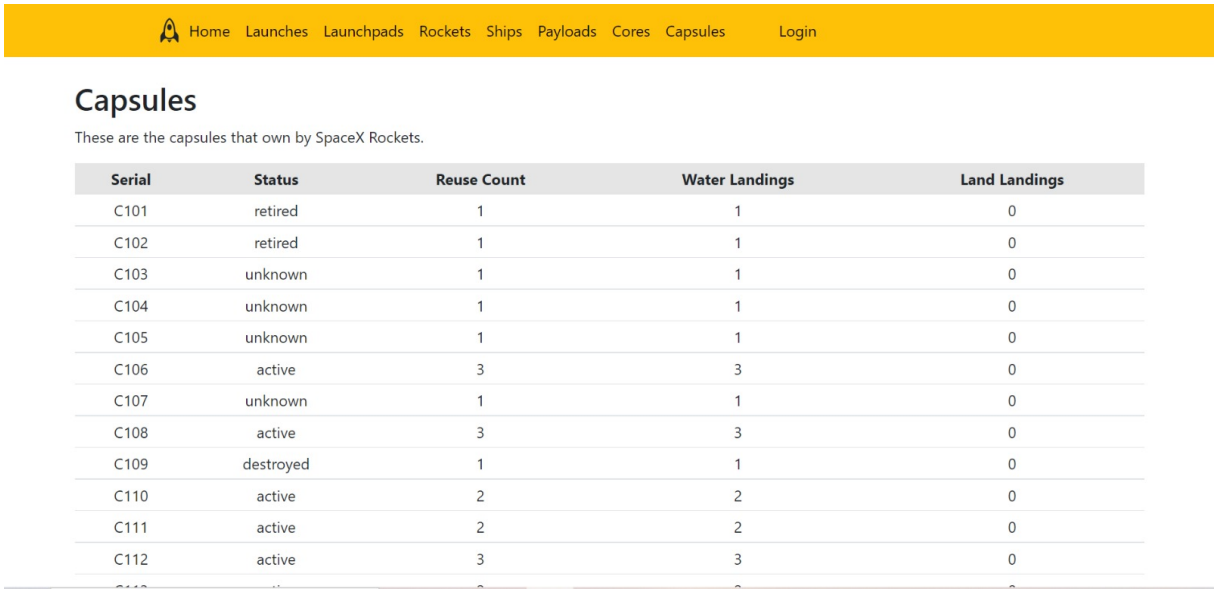


Figure 1: The ER Diagram of the SpaceX Database

1.1 Parts Owned by Hakkı Arda Çoha

1.1.1 Capsules Page

This page is designed to provide users with information about capsules in a read-only format. The Capsules page displays a table of capsule data from our database, allowing users to view various details about each capsule.Users do not have the ability to edit or modify the capsule data on this page.



Serial	Status	Reuse Count	Water Landings	Land Landings
C101	retired	1	1	0
C102	retired	1	1	0
C103	unknown	1	1	0
C104	unknown	1	1	0
C105	unknown	1	1	0
C106	active	3	3	0
C107	unknown	1	1	0
C108	active	3	3	0
C109	destroyed	1	1	0
C110	active	2	2	0
C111	active	2	2	0
C112	active	3	3	0

Figure 2: Capsules Page

1.1.2 Launches Page

All users of the application can access the Launches page to view information about launches and filter their search for specific launches. Users who are logged in to the application have the ability to add new launches, delete existing launches, or update the details of existing launches on this page. The Launches page is available to all users of the application.

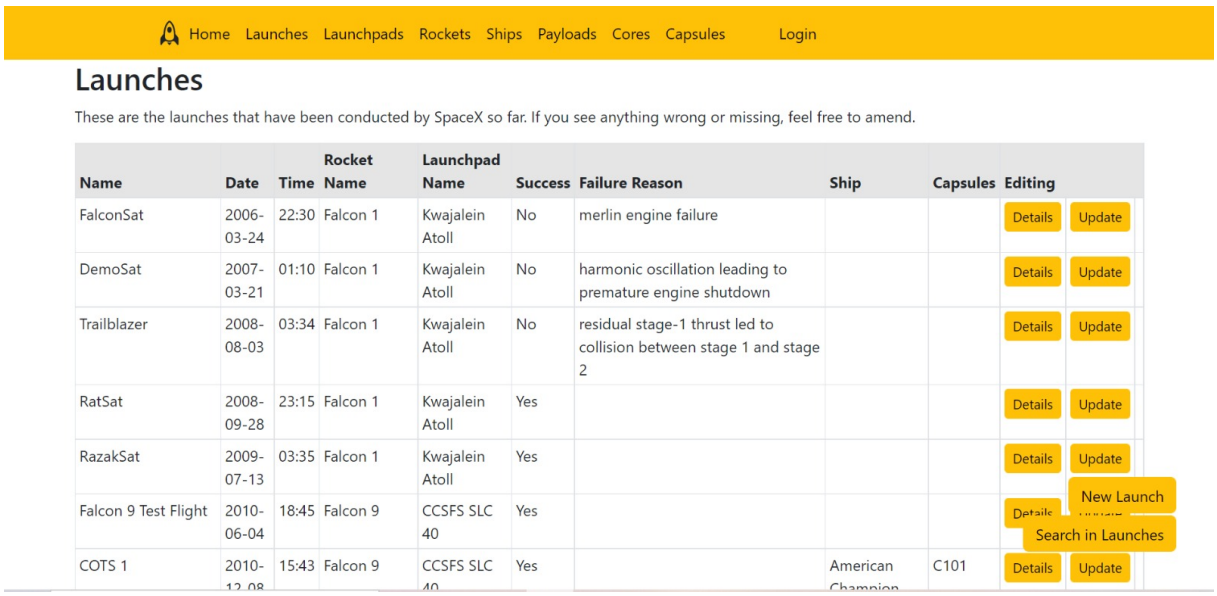


Figure 3: Launches Page

1.1.3 Launch Details Modal

Users can access the details of launches from clicking Details button which can be shown in Launches table. On that details modal, users can able to delete or update the details of Launches which kept in launch details table.

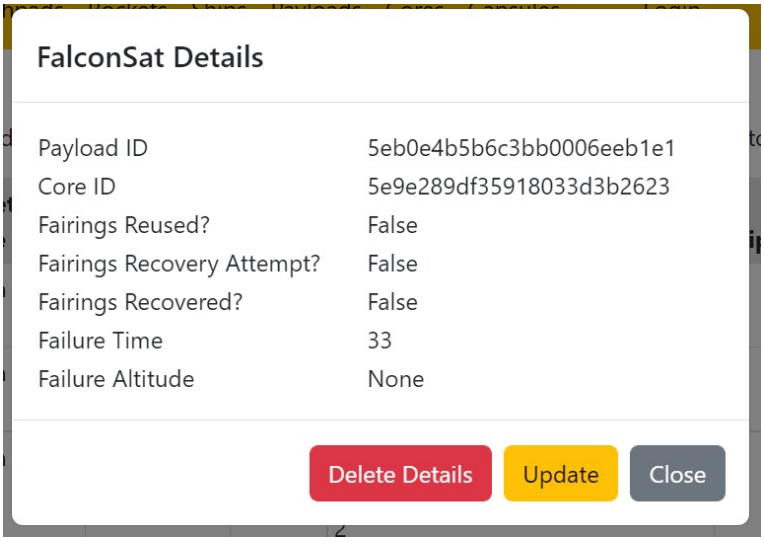


Figure 4: Launch Details

1.1.4 Add Launch

This feature allows users to contribute to our launches database by adding new launch data. Users can input various details about launches.After filling the relevant data, user can add to launches database by clicking add button in below. If user want to cancel that process, can easily click close button and cancel that process.

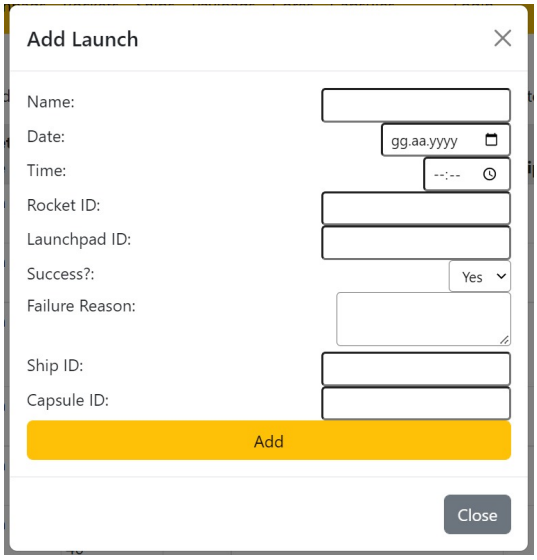


Figure 5: Adding data to Launches database

1.1.5 Delete Launch

This feature allows users to delete from launches database by clicking red ”Delete” button. User must be logged in. When clicked , this button deletes that relevant data from database.

1.1.6 Update Launch

In every row of the Launches Page, there is yellow Update button in Editing section which allows users to update the row they want.When users clicked the Update button, there is launch update form with relevant data from selected row. User updates the data they want after that , click the update button to successfully update the selected row.

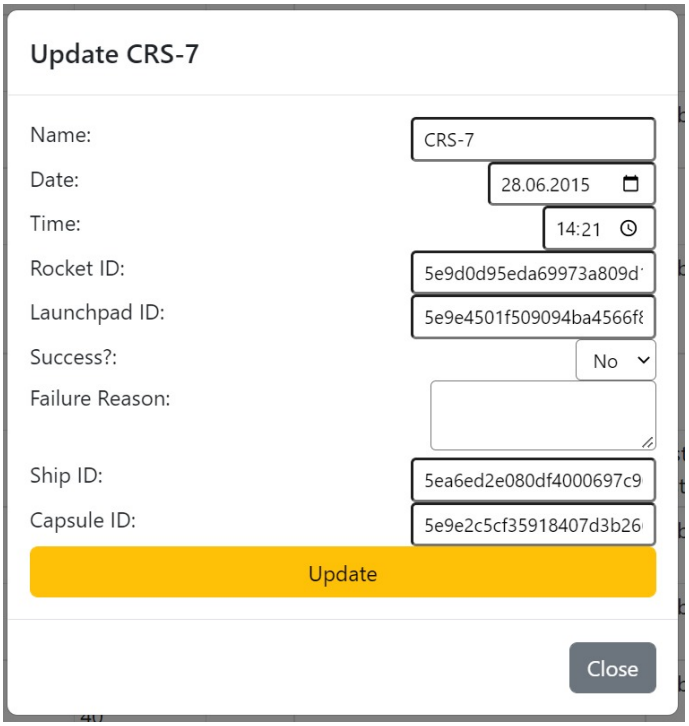


Figure 6: Update data from Launches database

1.1.7 Filter Launches

On the Launches page, users can search for specific launches by clicking the yellow ”Filter Launches” button at the bottom right of the page. This button will open the ”Launches Filtering Form,” which allows the user to fill in the required fields and initiate a search by clicking the yellow ”Filter” button. If the user decides not to search and wants to return to the Launches page, they can click the gray ”Close” button at the bottom of the form. This feature is available to all users, whether logged in to the application or not.

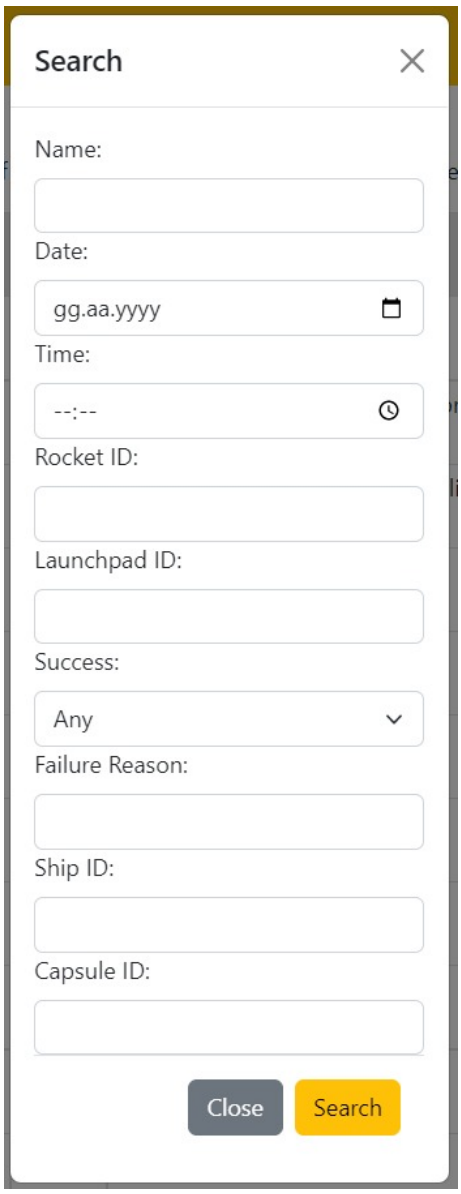


Figure 7: Filter data from Launches database

1.2 Parts Owned by Beyza Aydeniz

1.2.1 Engine Information Page

Users who want to access ”Rocket Details 1” data of rockets should click on the yellow ”Engine Info” button in the ”Details” column of the table on the Rockets page. Each rocket has its own ”rocket details 1” data. Users can access rocket details 1 data by clicking the ”Engine Info” button on the row of the relevant rockets.

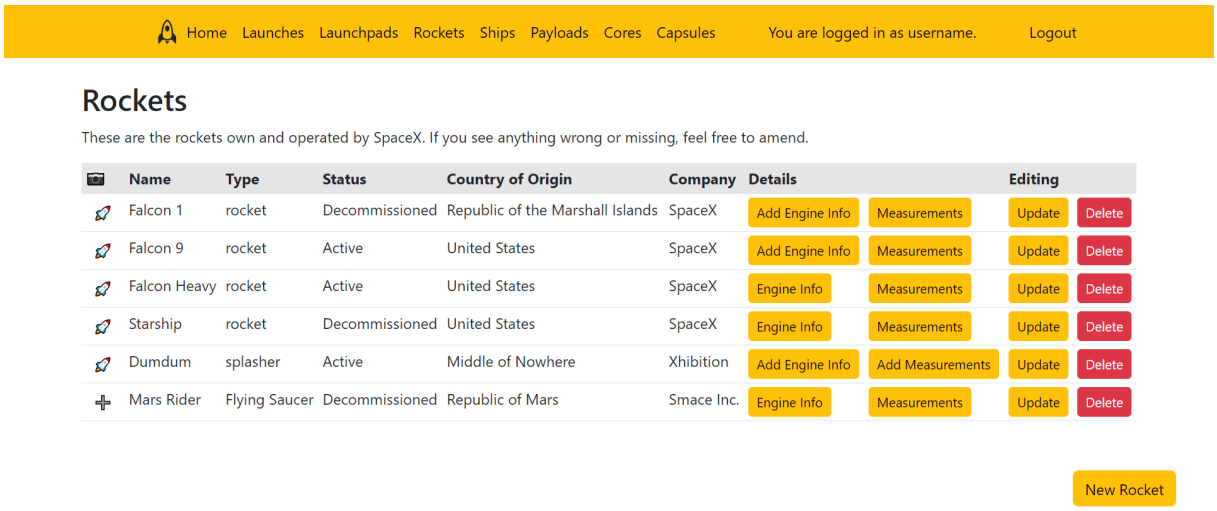


Figure 8: Engine Information page entered from Rockets page

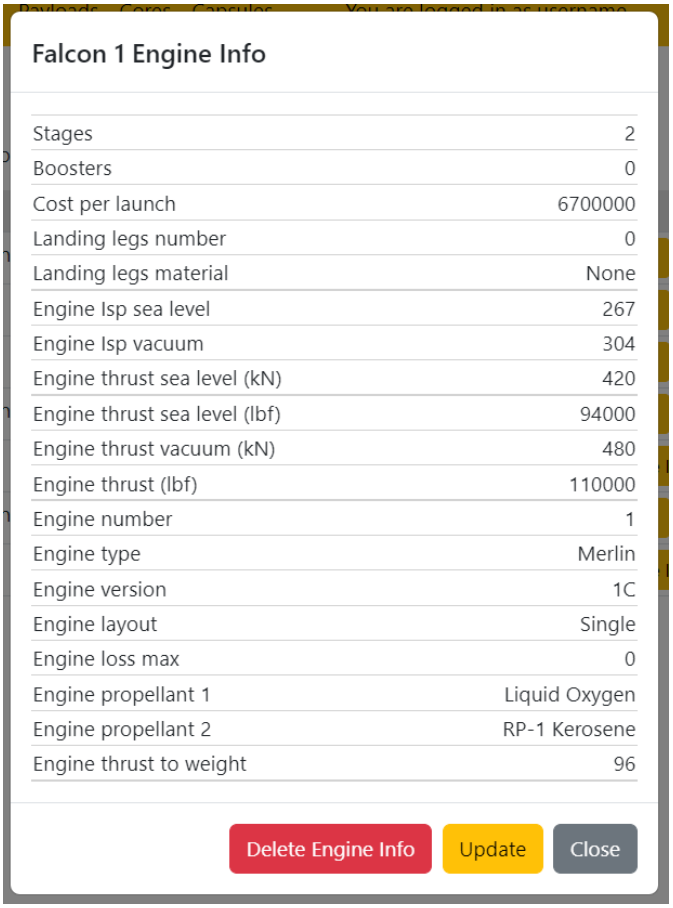


Figure 9: Engine Information page

1.2.2 Add Engine Information

In the table on the Rockets page, there is a yellow ”Add Engine Info” button in the ”Details” column in the relevant row of all rockets without Engine Info. When users press this button, ”Engine Info adding form” opens. Users who fill in the relevant fields can add a new engine information to the application by clicking the yellow ”Add” button at the bottom of the form. Users who change their decisions and do not want to add new engine information to the application can return to the rockets page by clicking the gray ”Close” button at the bottom right of the form.

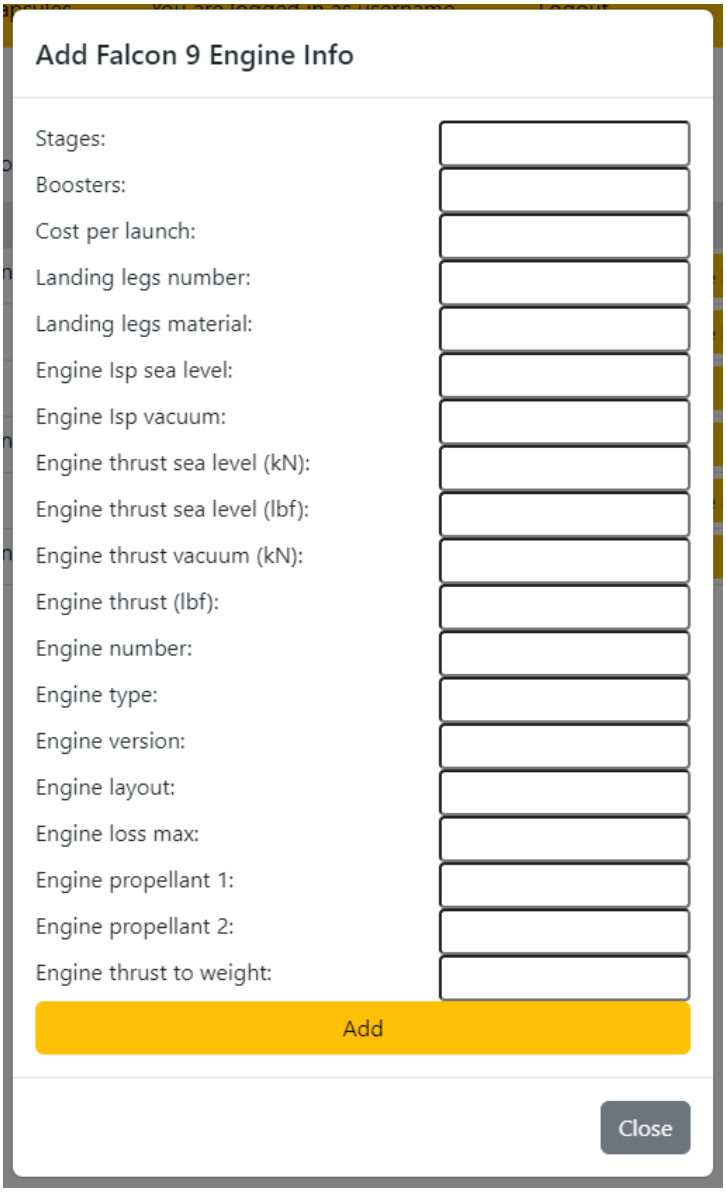


Figure 10: Engine Information Adding Form

1.2.3 Delete Engine Information

All rockets with engine information have a red button named ”Delete Engine Info” at the bottom of engine information page. Users can delete the engine information by clicking that button. After clicking the engine info delete button, the application automatically redirects users to the rockets page.

1.2.4 Update Engine Information

All rockets with engine info have a yellow button named ”Update” at the bottom of the “Engine Info” page. When users want to update their engine information and click on the ”update” button, the ”Engine Info update form” opens. The Engine Info update form is filled in on the users’ screen. Users who make the desired changes on the form can update the data in the application by clicking the yellow ”Update” button at the bottom of the form. Users who change their decisions and do not want to update rocket details can return to the rockets page by clicking the gray ”Close” button at the bottom right of the form.

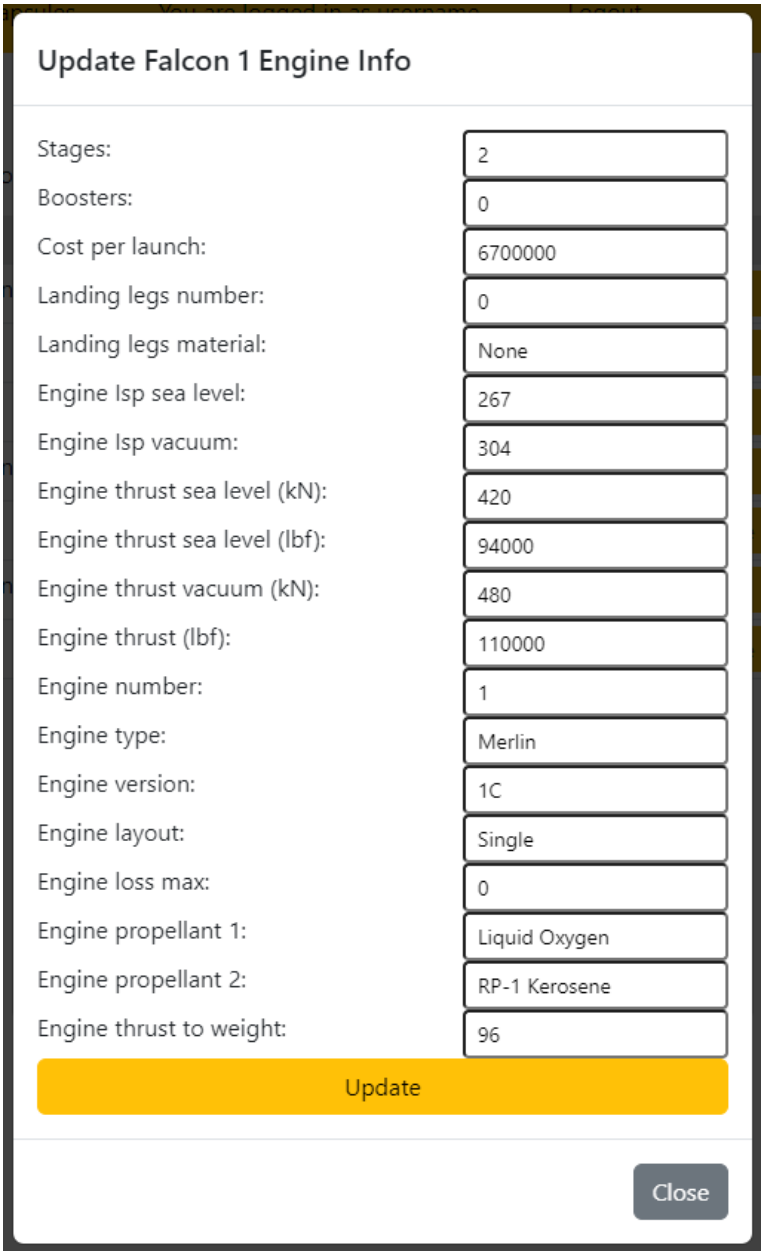


Figure 11: Engine Updating Form

1.2.5 Cores Page

Cores page allows all users of the application to access information about cores and search cores by filtering. Users who logged in to the application can add new cores to the table, delete cores in the table or update the details about the existing core.

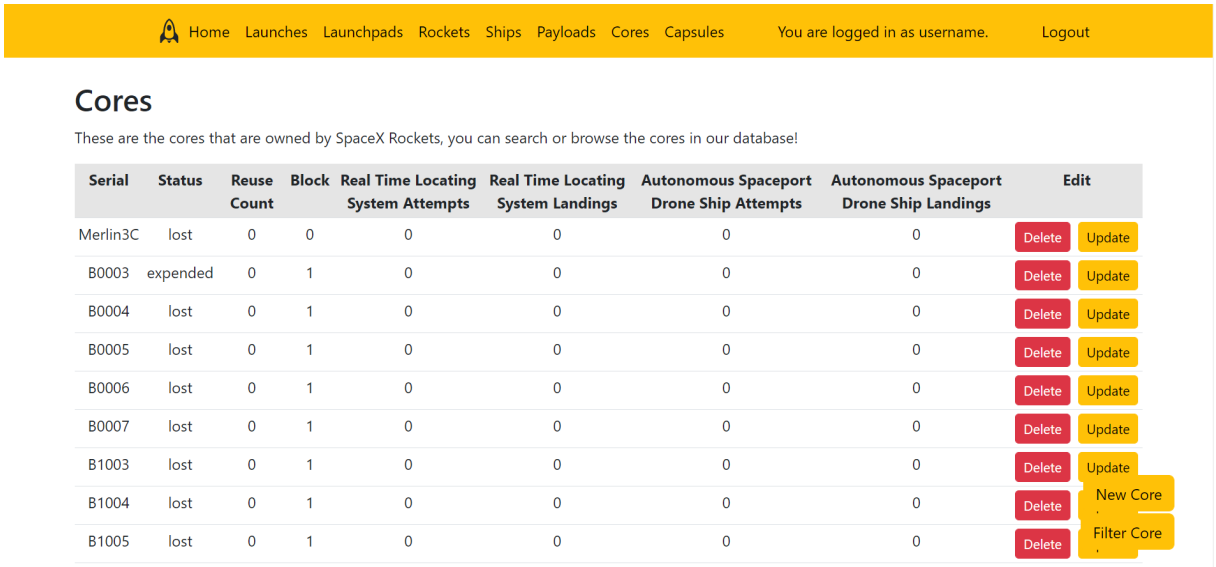


Figure 12: Cores Page

1.2.6 Add Core

When users logged in to the application click on the ”New Core” button fixed to the right bottom of the cores page, the ”core addition form” opens. Users who fill in the relevant fields can add a new core to the application by clicking the yellow ”Add” button at the bottom of the form. Users who change their decisions and do not want to add new cores to the application can return to the cores page by clicking the gray ”Close” button at the right bottom of the form. When users who are not logged in to the application press “New Core” button, the application automatically redirects the user to the ”login” page.

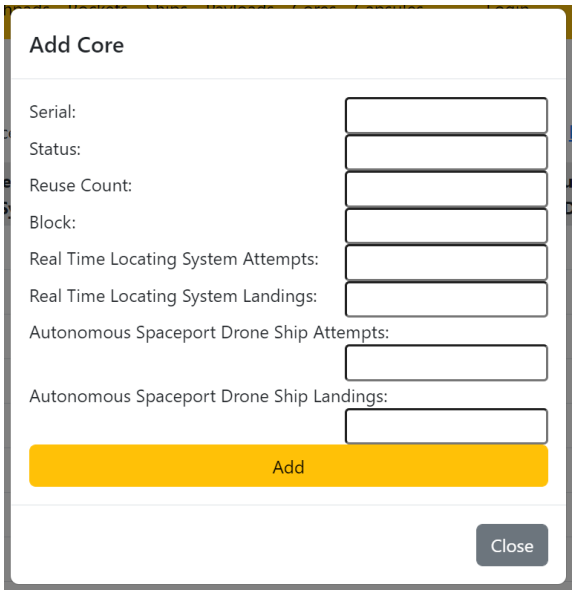
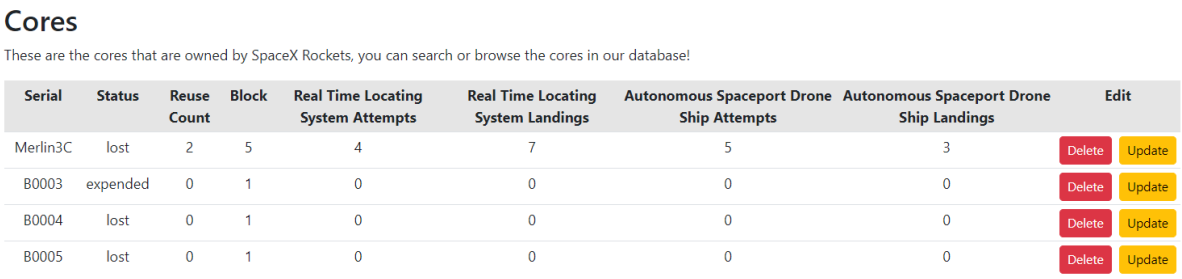


Figure 13: Core Addition Form

1.2.7 Delete Core

All cores have a red button named ”Delete” in the ”Editing” column of the table. Users logged in to the application can delete the core by clicking the ”Delete” button on the relevant core line. When users who are not logged in to the application press “Delete” button, the application automatically redirects the user to the ”login” page.



Cores

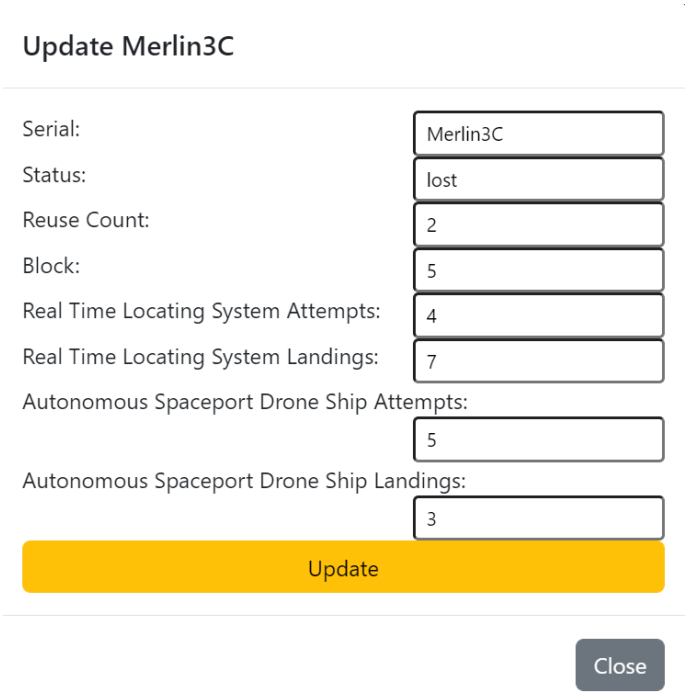
These are the cores that are owned by SpaceX Rockets, you can search or browse the cores in our database!

Serial	Status	Reuse Count	Block	Real Time Locating System Attempts	Real Time Locating System Landings	Autonomous Spaceport Drone Ship Attempts	Autonomous Spaceport Drone Ship Landings	Edit
Merlin3C	lost	2	5	4	7	5	3	Delete Update
B0003	expended	0	1	0	0	0	0	Delete Update
B0004	lost	0	1	0	0	0	0	Delete Update
B0005	lost	0	1	0	0	0	0	Delete Update

Figure 14: Cores with Delete Button

1.2.8 Update Core

All cores have a yellow button named ”Update” in the ”Editing” column of the table. When users logged in to the application, who want to update their core information, click on the ”update” button on the relevant line, the ”core update form” opens. When users who are not logged in to the application press “Update” button, the application automatically redirects the user to the ”login” page. The Core update form is filled in on the users’ screen. Users who make the desired changes on the form can update the data in the application by pressing the yellow ”Update” button at the bottom of the form. Users who change their decisions and do not want to update core details can return to the cores page by clicking the gray ”Close” button at the bottom right of the form.



Update Merlin3C

Serial: Merlin3C

Status: lost

Reuse Count: 2

Block: 5

Real Time Locating System Attempts: 4

Real Time Locating System Landings: 7

Autonomous Spaceport Drone Ship Attempts: 5

Autonomous Spaceport Drone Ship Landings: 3

Update

Close

Figure 15: Cores Updating Form

1.2.9 Filter Core

All users who have logged in to the application or not, can search the cores page by clicking the yellow ”Filter Core” button pinned to the bottom right of the page. When the user clicks on the Filter core button, the ”Core filtering form” opens. The user, who fills in the required parts of the form, can search by clicking the yellow ”Filter” button at the right bottom of the form. Users who change their decisions and do not want to search cores can return to the cores page by clicking the gray ”Close” button at the bottom of the form.

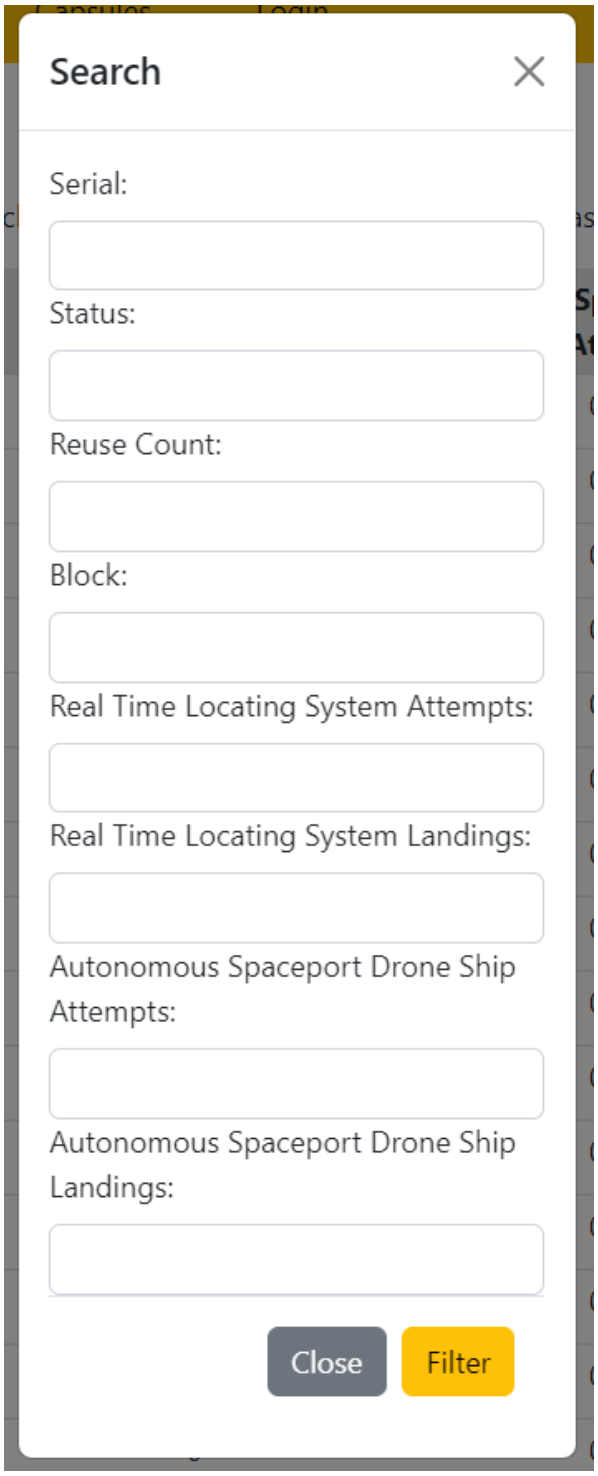


Figure 16: Cores Filtering Form

1.3 Parts Owned by Abdullah Asım Emül

1.3.1 Rockets (Main)

Rockets (Main) Viewing Page Users who wish to browse the rockets recorded in the system can do so from the "Rockets" page. Users can learn about rockets' names, types, statuses, countries of origin, and companies. (Figure 17)

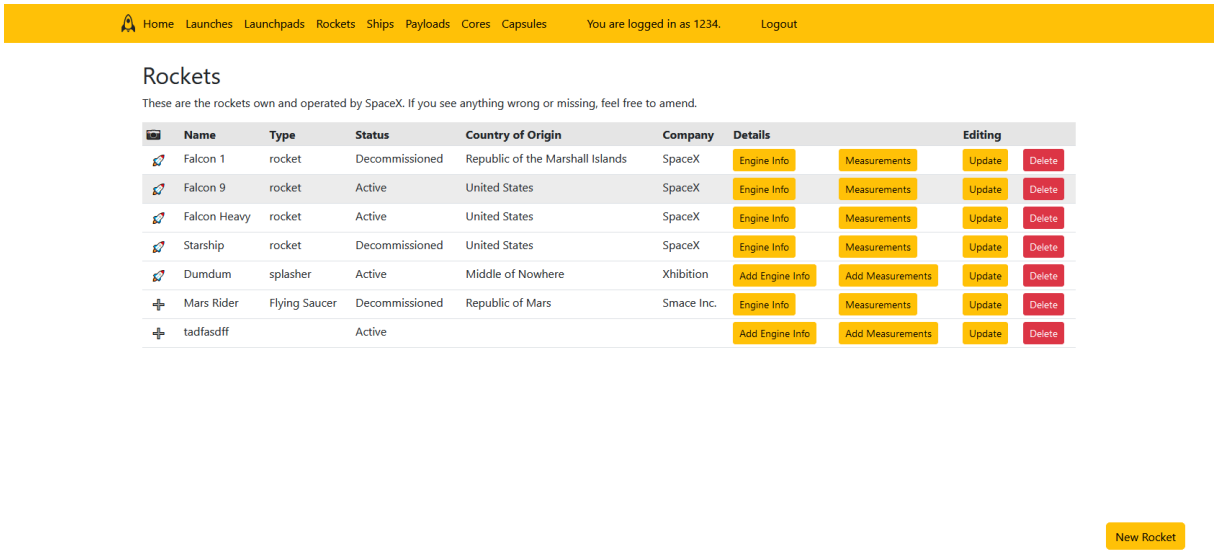


Figure 17: Main rockets view

Rockets (Main) Adding Modal Users who wish to add a rocket to the system may do so via the "New Rocket" button at the bottom right of the rockets viewing page.

After clicking this button a modal will appear allowing the user to enter the information of the rocket they wish to add. This information is the same as the one displayed on the rockets viewing page. Once all desired information has been entered, the "Add" button will add the information to the system. (Figure 18)

Rockets (Main) Editing Modal Users who wish to edit a rocket's information may do so by clicking "Update" button of the relevant rocket. A modal will appear with the current information of that rocket filled in. This information may be changed and the "Update" button pressed to save the changes to the system. (Figure 19)

Rocket (Main) Deletion If a user wishes to delete a rocket saved to the system, they may do so if they are logged in and an admin. If this is the case, a "Delete" button will appear on the row of each rocket, upon clicking which will cause that rocket and its other details to be deleted from the system. (Figure 17)

1.3.2 Rocket Measurements

Rocket Measurements Viewing Page Users who wish to view the height, diameter, and mass measurements in metric and imperial units of a rocket may click the "Measurements" button of the relevant rocket. It is important to note that this button will only appear if this information exists for the rocket in question. This will display a modal with the aforementioned

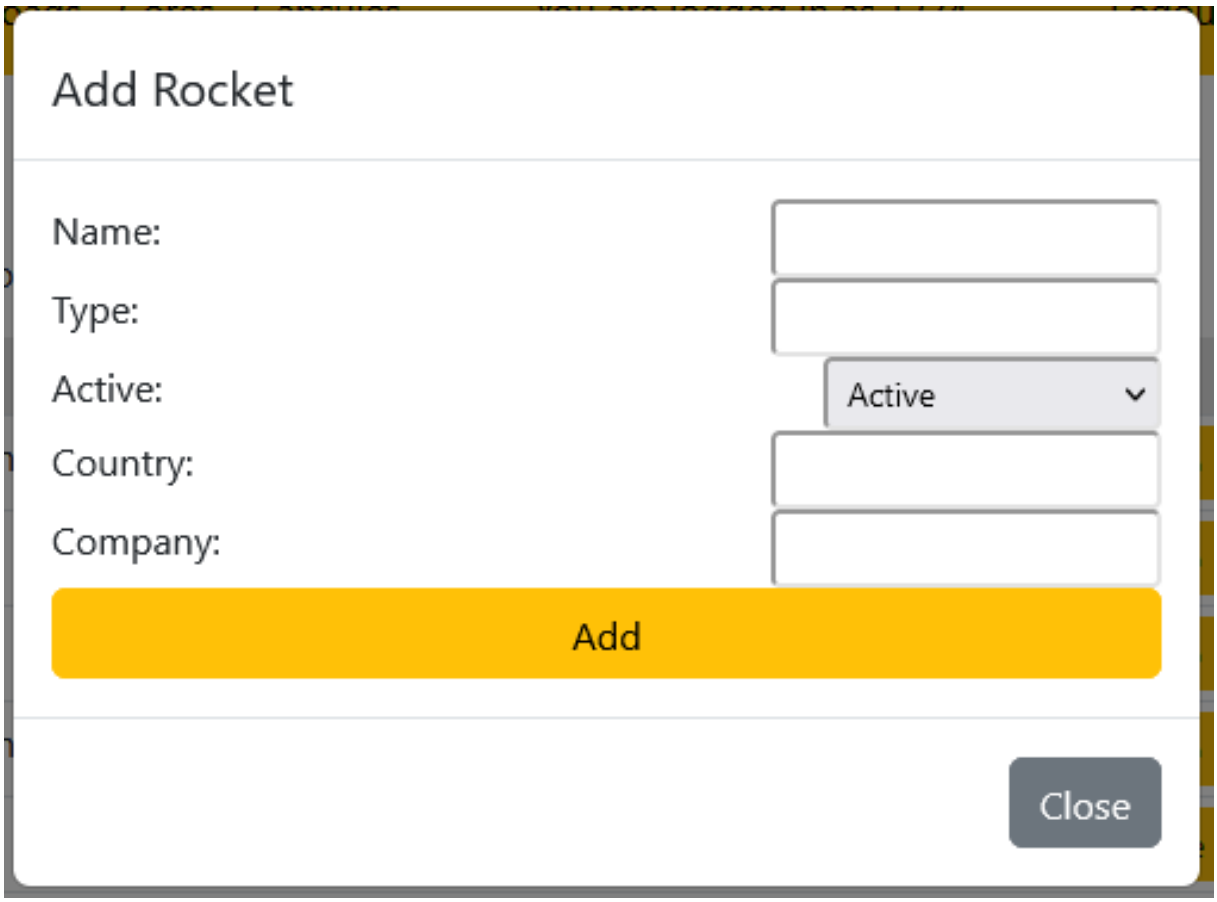


Figure 18: Main rockets adding view

information. The height and diameter values can be decimal values while the mass values may only be integers. (Figure 20)

Rocket Measurements Adding Modal Rockets with no measurement information will have an "Add Measurements" button instead of a "Measurements" button. Clicking this button will present the user with a modal where measurement information may be entered.

It is advised that information entered for a given measurement's value in metric be double-checked against its value in imperial and vice versa.

Once all information has been entered and checked, clicking the "Add" button will save this information to permanent storage. (Figure 21)

Rocket Measurements Editing Modal Users who wish to edit the measurements of a rocket must first open the measurements modal of the rocket they wish to edit then click on the "Update" button at the bottom of the modal. This will open a new modal where the user can edit the desired information and save this via the "Update" button of this modal. (Figure 22)

Rocket Measurements Deletion If a user wishes to delete the measurements of a rocket, they may click the "Delete Measurements" at the bottom of the measurements modal. (Figure 17)

1.3.3 Rocket Images

Rocket Image Viewing Page Users who wish to view the image of a rocket may press the rocket emoji on the row of the rocket they wish to view the image of. This button will appear only for rockets with image information present in the system. (Figure 23)

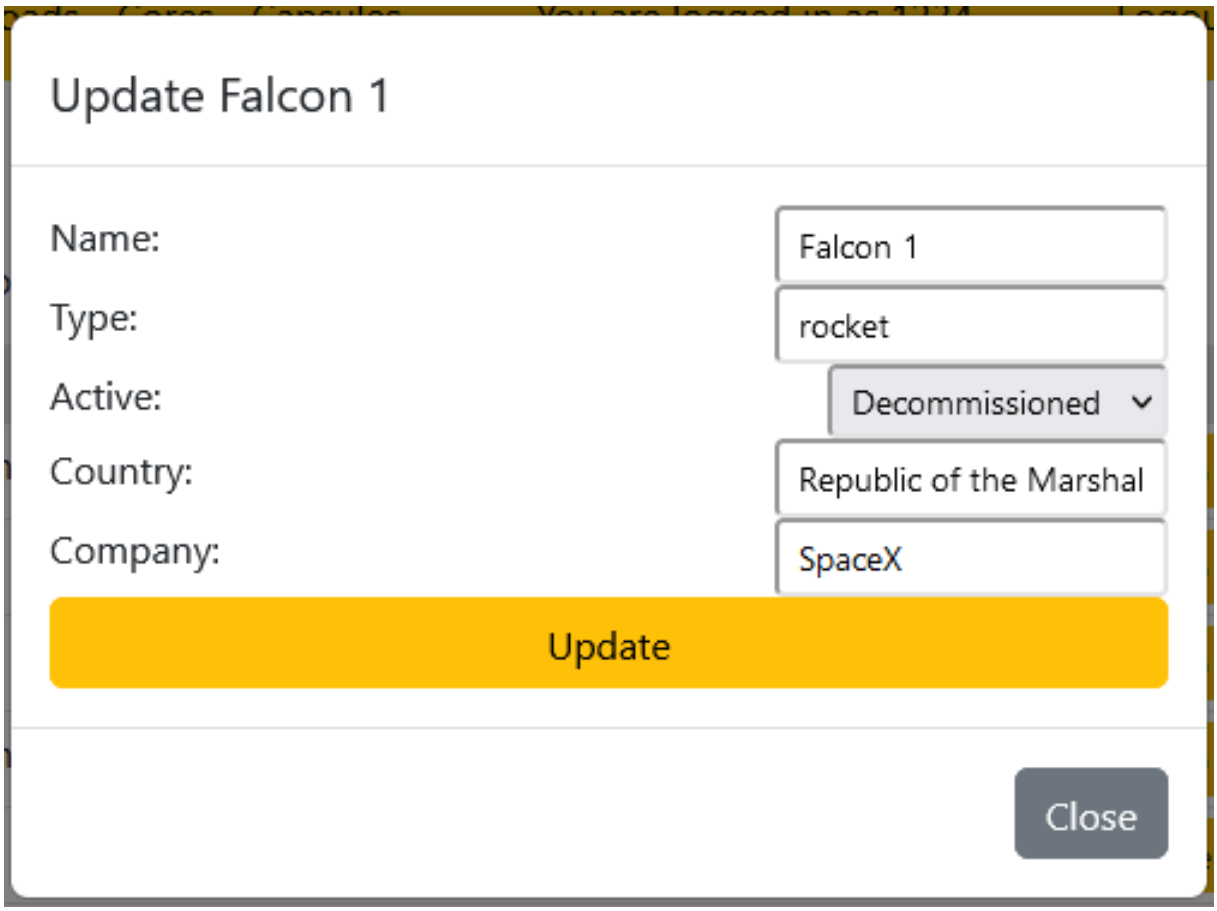


Figure 19: Main rockets editing view

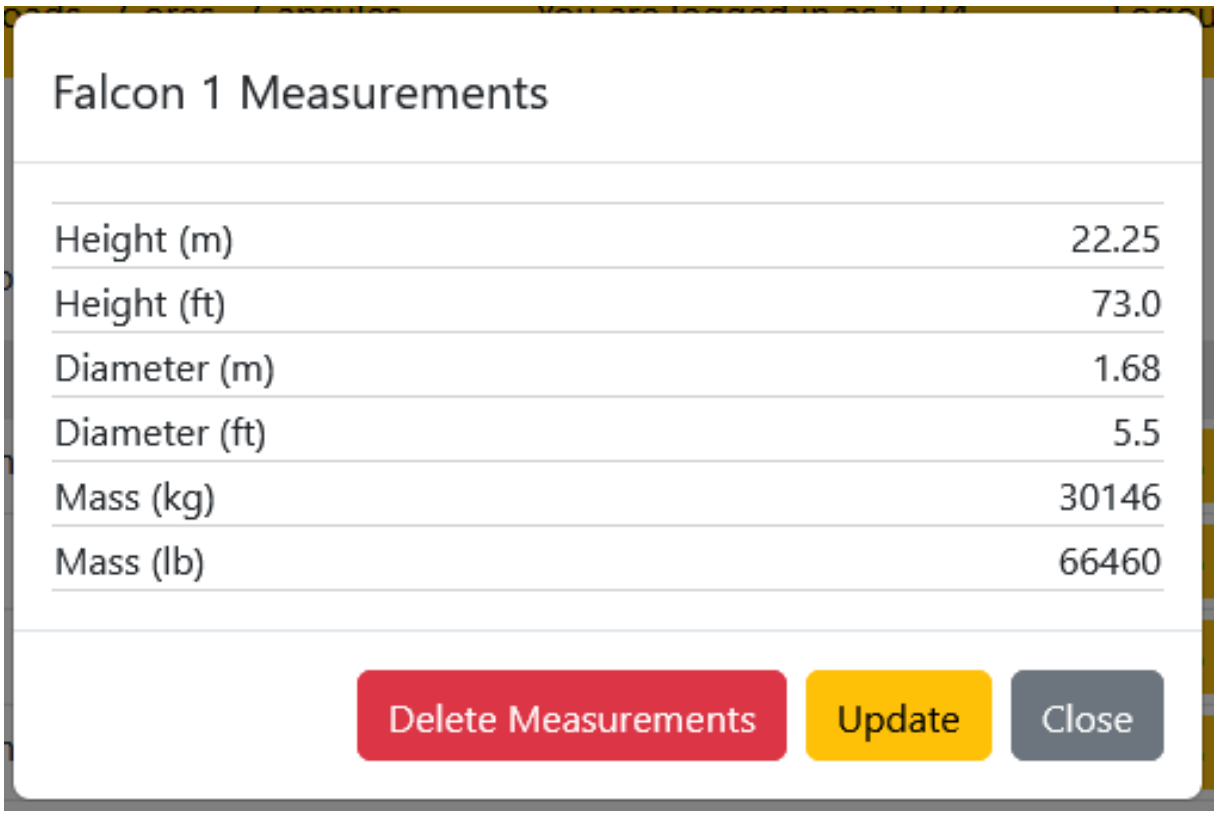


Figure 20: Rocket measurements view

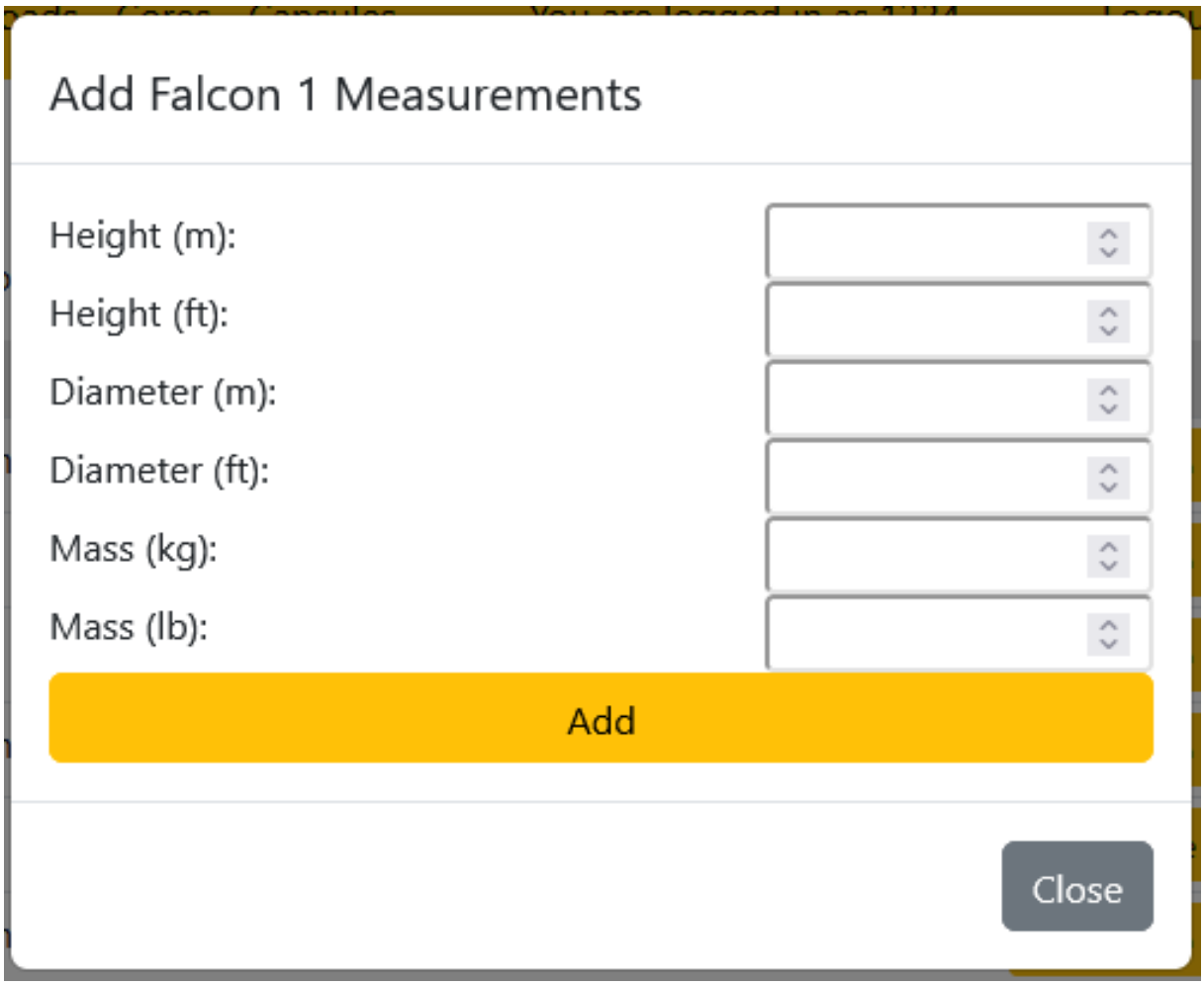


Figure 21: Rocket measurements adding view

Rocket Image Adding Modal Users who wish to add an image to a rocket without one may do so by pressing the plus emoji on the row of the relevant rocket. This will bring up a modal with a file upload input where the desired image may be selected. Clicking the "Add" button will add the selected to the system. (Figure 24)

Rocket Image Deletion If a user wishes to delete the image of a rocket, they may click the "Delete Image" button at the bottom of the relevant rocket image viewing modal. (Figure 23)

1.3.4 Launchpads Viewing Page

Users who wish to view the launchpads owned by SpaceX may open the "Launchpads" page from the navigation bar at the top of the website. This page contains the short name, full name, status, locality, region, latitude, longitude, and timezone of a given launchpad. (Figure 25)

1.3.5 Signup Page

Users who do not have an account and would like to register may do so via the signup page accessible through the prompt present in the login page.

In this page users can specify a username and password, the former of which must not be taken by a prior user. They must also state whether they are a superfan of SpaceX or not along with whether they wish to be considered for adminhood. After filling in all these fields, users

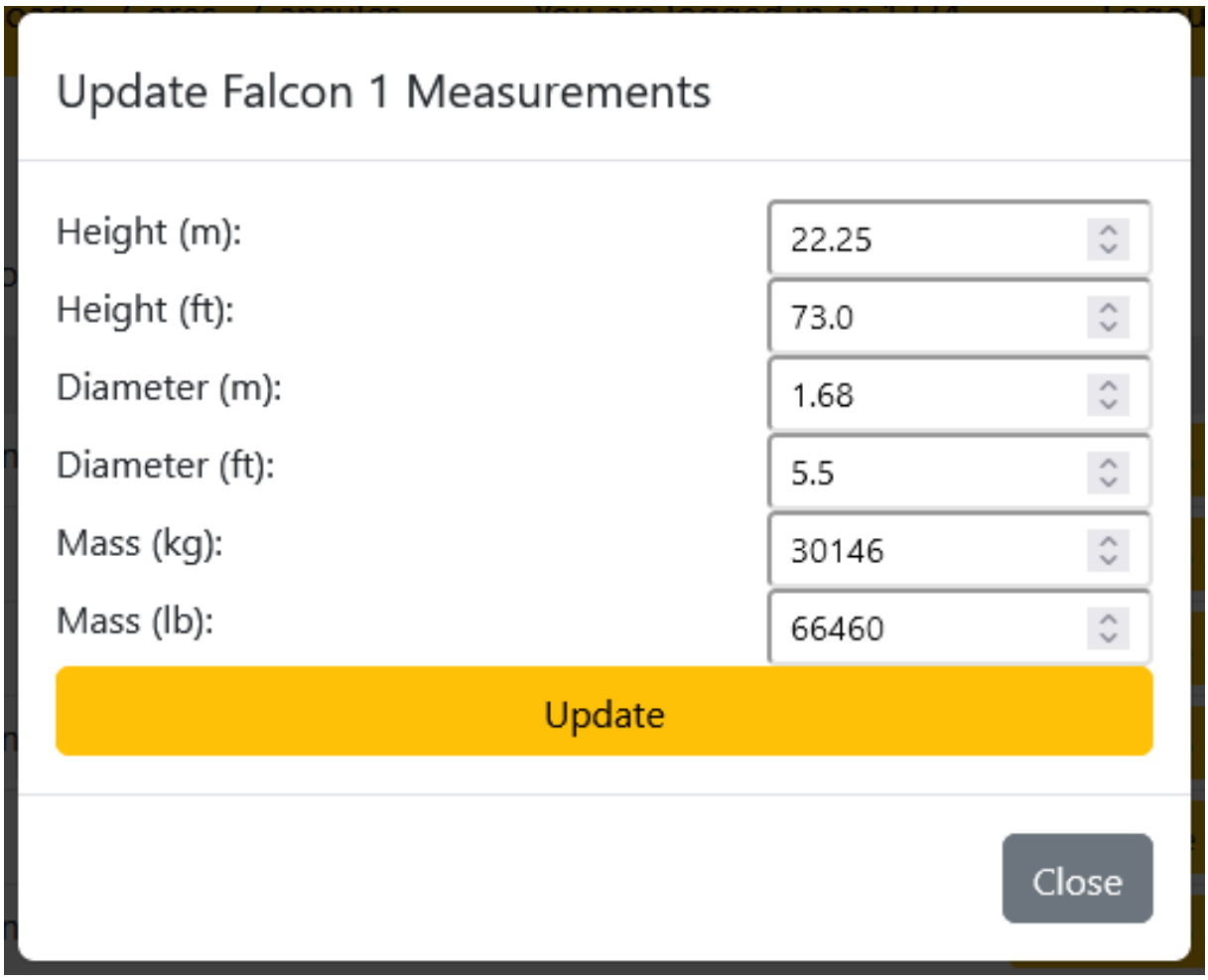


Figure 22: Rockets measurements editing view

can press the “Sign up” button to create their account. Doing so will take users to the login page. (Figure 26)

1.4 Parts Owned by Alp Türkbayrak

1.4.1 Base HTML

All of the HTML files extend ”base.html”. This is done by using Jinja2 Templating Language and using ”extends” keyword. This is the template of our app’s page designs. The navbar helps users navigate inside of our app while looking stylish.(Figure 27)

1.4.2 Home Page

home.html extends base.html like the other pages. It utilises basic cards to get the user started with our app. (Figure 28)

1.4.3 Ship Technics

Ship Technics are information in the ship_details_1 table. They show up as a button that controls a modal in the ships page of our app.(Figure 29)

When clicked, a modal pops up. It displays information if it has any in the table. If the attribute is NULL, it displays a ’-’ sign.(Figure 30)

The user can delete the information by clicking the red button, or they can update it by using the yellow button.(Figure 31)

1.4.4 Payloads Information Page

Payloads page allows all users of the application to access information about payloads and search payloads by filtering. Users who logged in to the application can add new payloads to the table, delete payloads in the table or update the information about the existing payloads.(Figure 32)

Add Payload: The ”New Payload” button fixed to the lower right corner of the screen is used to insert new rows to the payloads table of our database. When clicked, the payload addition form modal appears.(Figure 33)

If the user is not logged in when ”Add” is clicked, the application automatically redirects the user to the ”login” page.

Delete Payload: If the user is logged in, a red button that says ”Delete” appears at the end of each row. When clicked, this button simply deletes the row from the database.(Figure 34)

Update Payload: If the user thinks an information presented is wrong or outdated, they can update it using the yellow ’Update’ button at the end each row. When clicked, this button opens a form modal. The user can change the information in this form to update the desired row.(Figure 35)

Search Payloads: If the user only wants the app to display certain payloads, they can do so by using the ”Search in Payloads” button fixed to the bottom right side of the screen. When clicked, a modal pops up displaying the parameters the user can search with. If any of the form elements are left blank, the app will not take them into account while filtering.(Figure 36)

1.5 Parts Owned by Ahmet Metehan Yaman

1.5.1 User Login Page

First of all, We designed a user login page with a simple html page. We provided the transitions to login to our Flask application with this page. At first, user information and passwords were kept in a txt file. And this is how user validation was provided. But this was not useful and reliable. If the user entered the username and password same with in the txt file, he could log into the application. Then, a new userlogin page was designed using the flask-login library. You can see in Figure 37.

In this way, user information can be kept in a database. Users can also subscribe to our application. While becoming a member, they can also get admin authority if they want. When users become members, their usernames and passwords are registered in our database. While saving passwords, we create an inaccessible and secure system by hashing the passwords. We are using passlib library for that.

1.5.2 Ships Page

The application's Ships page enables all users to view ship-related data and conduct filter-based ship searches. Users who have logged in to the program can change the information about the existing ships or add new ships to the table.You can see in Figure 38.

Add Ship: The ”New ship” button fixed to the lower right corner of the screen is used to insert new rows to the ships table of our database. When clicked, the ship addition form modal appears.You can see in Figure 39.

If the user is not logged in when ”Add” is clicked, the application automatically redirects the user to the ”login” page.

Delete Ships: If the user is logged in, a red button that says ”Delete” appears at the end of each row. When clicked, this button simply deletes the row from the database.You can see in Figure 40.

Update ship: If the user thinks an information presented is wrong or outdated, they can update it using the yellow 'Update' button at the end each row. When clicked, this button opens a form modal. The user can change the information in this form to update the desired row.You can see in Figure 41.

Search ships: If the user only wants the app to display filtered ships, they can do so by using the ”Search in ships” button fixed to the bottom right side of the screen. When clicked, a modal pops up displaying the parameters the user can search with. If any of the form elements are left blank, the app will not take them into account while filtering.You can see in Figure 42.

1.5.3 Ships Measurements

Ship Measurement are information in the ship_details_2 table. They show up as a button that controls a modal in the ships page of our app. If you press the measurement button on the ships page you can open a modal to see the measurements info.You can see in Figure 43.

You can also add, delete, and update measurements.

Add measurement: The ”Add measurement” button provide to add new measurement info.You can see in Figure 44 and Figure 45.

If the user is not logged in when ”Add” is clicked, the application automatically redirects the user to the ”login” page.

Delete measurements: If the user is logged in, a red button that says ”Delete” appears at the end of the modal. When clicked, this button simply deletes the measurement info from the database.You can see in Figure 46.

Update measurements: If the user thinks an information presented is wrong or outdated, they can update it using the yellow ’Update’ button at the end each row. When clicked, this button opens a form modal. The user can change the information in this form to update the desired row.You can see in Figure 47.

2 DEVELOPER GUIDE

2.1 Parts Owned by Hakkı Arda Çoha

2.1.1 Launches Page

The view function parts for launches in views.py:

```
@views.route('/launches', methods=['GET', 'POST'])
def launches():
    launch_data = database.get_launches()
    launch_details = database.get_launch_details()

    inexistent_launch_detail = []
    inexistent_launch_detail_dict = []

    rocket_data = database.get_rockets()
    rocket_dict = {}
    for rocket in rocket_data:
        rocket_dict[rocket["rocket_id"]] = rocket["name"].replace(" ", " ")

    launchpad_data = database.get_launchpads()
    launchpad_dict = {}
    for launchpad in launchpad_data:
        launchpad_dict[launchpad["launchpad_id"]] = launchpad["name"]

    ship_data = database.get_ships_only()
    ship_dict = {}
    for ship in ship_data:
        ship_dict[ship["ship_id"]] = ship["name"]

    capsule_data = database.get_capsules()
    capsule_dict = {}
    for capsule in capsule_data:
        capsule_dict[capsule["capsule_id"]] = capsule["serial"]

    present = False
    for launch in launch_data:
        for launch_detail in launch_details:
            if launch["launch_id"] == launch_detail["launch_id"]:
                present = True
                break
        if not present:
            inexistent_launch_detail += [launch["launch_id"]]
            inexistent_launch_detail_dict += [{"launch_id": launch["launch_id"], "name": launch["name"]}]
        present = False

    launch_data = database.get_launches()
    return render_template("launches.html", launches=launch_data, launch_details = launch_details,
    inexistent_launch_detail=inexistent_launch_detail, inexistent_launch_detail_dict=inexistent_launch_detail_dict,
    formM=forms.LaunchForm(), formD=forms.LaunchDetailForm(),
    rockets=rocket_dict, launchpads = launchpad_dict, ships = ship_dict, capsules = capsule_dict)
```

SQL statement in database.py:

```
def get_launches():
    with sqlite3.connect(db_location) as con:
```

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            "SELECT * FROM launches LEFT JOIN launch_details on launches.
    launch_id = launch_details.launch_id"
            ).fetchall()
```

```
def get_launch_details():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            """SELECT * FROM launch_details
            JOIN (SELECT launch_id, name FROM launches) AS launch_names
            ON launch_details.launch_id = launch_names.launch_id"""
            ).fetchall()
```

2.1.2 Capsules Page

The view function parts for capsules which is read only page in views.py:

```
@views.route('/capsules', methods=['GET', 'POST'])
def capsules():
    capsule_data = database.get_capsules()
    return render_template("capsules.html", capsules=capsule_data)
```

SQL statement in database.py:

```
def get_capsules():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            "SELECT * FROM capsules"
            ).fetchall()
```

2.1.3 Launch Adding

The view function parts for adding launches in views.py:

```
@views.route("/add_launch", methods=["POST"])
def add_launch():
    database.add_launch(request)
    return redirect(url_for("views.launches"))
```

SQL statement in database.py:

```
def add_launch(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new launch
        launch_columns = cur.execute("PRAGMA table_info(launches)").
    fetchall()

        new_launch = [str(current_app.config["launch_id"])]
        current_app.config["launch_id"] += 1
        for column in launch_columns:
            if column["name"] in request.form.keys():
                new_launch += [request.form[column["name"]]]
```

```

        cur.execute(f'INSERT INTO launches VALUES ({",".join("?" * len(launch_columns))})', new_launch)
        con.commit()
```

2.1.4 Launch Deleting

The view function parts for deleting launches in views.py:

```
@views.route('/delete_launch', methods=['GET'])
def delete_launch():
    database.delete_launch(request.args.get("launch_id"))
    return redirect(url_for("views.launches"))
```

SQL statement in database.py:

```
def delete_launch(launch_id):
        with sqlite3.connect(db_location) as con:
            cursor = con.cursor()
            cursor.execute("PRAGMA foreign_keys=ON")  # This allows for cascading to details
            query = "DELETE FROM launches WHERE (launch_id = ?)"
            cursor.execute(query, (launch_id,))
            con.commit()
```

2.1.5 Launch Updating

The view function parts for updating launches in views.py:

```
@views.route('/update_launch', methods=['POST'])
def update_launch():
    database.update_launch(request)
    return redirect(url_for("views.launches"))
```

SQL statement in database.py:

```
def update_launch(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update launch
        launch_columns = cur.execute("PRAGMA table_info(launches)").fetchall()
        launch_columns_str = ",".join([column["name"] + "=?" for column in launch_columns if column["name"] != "launch_id"])

        launch = []
        for column in launch_columns:
            if column["name"] in request.form.keys():
                launch += [request.form[column["name"]]]

        launch_id = launch[0]
        launch = launch[1:] + [launch_id]

        cur.execute(f'UPDATE launches SET {launch_columns_str} WHERE launch_id = ?', launch)
        con.commit()
```

2.1.6 Launch Filtering

The view function parts for filtering launches in views.py:

```
@views.route('/launches_filtered', methods=['GET', 'POST'])
def launches_filtered():
    filter_data = database.filter_launches(request)
    return render_template("launches.html", launches=filter_data, formM=
   forms.LaunchForm())
```

SQL statement in database.py:

```
def filter_launches(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        print("filter_launches executed")
        query = "SELECT * FROM launches"
        params = []
        print(request.form.get('fname'))
        if request.form.get('fname'):
            if not params:
                query += " WHERE"
            user_name = request.form['fname']
            query += " name LIKE ?"
            param_name = "%" + user_name + "%"
            params.append(param_name)

        if request.form.get('fdate'):
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_date = request.form['fdate']
            query += " date LIKE ?"
            param_date = "%" + user_date + "%"
            params.append(param_date)

        if request.form.get('ftime') != "":
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_time = request.form['ftime']
            query += " time LIKE ?"
            param_time = "%" + user_time + "%"
            params.append(param_time)

        if request.form.get('frocket_id'):
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_rocket_id = request.form['frocket_id']
            query += " rocket_id LIKE ?"
            param_rocket_id = "%" + user_rocket_id + "%"
            params.append(param_rocket_id)

        if request.form.get('flaunchpad_id'):
            if not params:
                query += " WHERE"
            else:
```

```
                query += " AND"
            user_launchpad_id = request.form['flaunchpad_id']
            query += " launchpad_id LIKE ?"
            param_launchpad_id = "%" + user_launchpad_id + "%"
            params.append(param_launchpad_id)

        if request.form.get('fsuccess'):
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_success = request.form['fsuccess']
            query += " success LIKE ?"
            param_success = "%" + user_success + "%"
            params.append(param_success)

        if request.form.get('ffailure_reason'):
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_failure_reason = request.form['ffailure_reason']
            query += " failure_reason LIKE ?"
            param_failure_reason = "%" + user_failure_reason + "%"
            params.append(param_failure_reason)

        if request.form.get('fship'):
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_ship = request.form['fship']
            query += " ship LIKE ?"
            param_ship = "%" + user_ship + "%"
            params.append(param_ship)

        if request.form.get('fcapsules'):
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_capsules = request.form['fcapsules']
            query += " capsules LIKE ?"
            param_capsules = "%" + user_capsules + "%"
            params.append(param_capsules)

        print(query)
        print(tuple(params))
        filter = cur.execute(query, tuple(params)).fetchall()
        return filter
```

2.1.7 Launch Details Adding

The view function parts for adding launch details in views.py:

```
@views.route('/add_launch_detail', methods=['POST'])
def add_launch_detail():
    database.add_launch_details(request)
    return redirect(url_for("views.launches"))
```

SQL statement in database.py:

```python
def add_launch_details(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new launch_details
        launch_details_columns = cur.execute("PRAGMA table_info(launch_details)").fetchall()

        new_launch_details = []
        for column in launch_details_columns:
            if column["name"] in request.form.keys():
                new_launch_details += [request.form[column["name"]]]

        cur.execute(f'INSERT INTO launch_details VALUES ({",".join("?" * len(launch_details_columns))})', new_launch_details)
        con.commit()
```

2.1.8 Launch Details Deleting

The view function parts for deleting launch details in views.py:

```python
@views.route('/delete_launch_detail', methods=['GET'])
def delete_launch_detail():
    database.delete_launch_details(request.args.get("launch_id"))
    return redirect(url_for("views.launches"))
```

SQL statement in database.py:

```python
def delete_launch_details(launch_id):
    with sqlite3.connect(db_location) as con:
        cursor = con.cursor()
        query = "DELETE FROM launch_details WHERE (launch_id = ?)"
        cursor.execute(query, (launch_id,))
        con.commit()
```

2.1.9 Launch Details Updating

The view function parts for updating launch details in views.py:

```python
@views.route('/update_launch_detail', methods=['POST'])
def update_launch_detail():
    database.update_launch_detail(request)
    return redirect(url_for("views.launches"))
```

SQL statement in database.py:

```python
def update_launch_detail(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update launch_detail
        launch_detail_columns = cur.execute("PRAGMA table_info(launch_details)").fetchall()
        launch_detail_str = ",".join([column["name"] + "=?" for column in launch_detail_columns if column["name"] != "launch_id"])

        launch_detail = []
        for column in launch_detail_columns:
            if column["name"] in request.form.keys():
```

```
                launch_detail += [request.form[column["name"]]]

        launch_id = launch_detail[0]
        launch_detail = launch_detail[1:] + [launch_id]

        cur.execute(f'UPDATE launch_details SET {launch_detail_str} WHERE launch_id = ?', launch_detail)
        con.commit()
```

2.1.10 Form Fields

Wtforms library used for form fields. The form field parts for launches in forms.py:

```
class LaunchForm(FlaskForm):
    launch_id = StringField()
    name = StringField()
    date = DateField()
    time = TimeField()
    rocket_id = StringField()
    launchpad_id = StringField()
    success = SelectField(choices=[("True", "Yes"), ("False", "No")])
    failure_reason = TextAreaField()
    ship = StringField()
    capsules = StringField()
```

The form field parts for launch details in forms.py:

```
class LaunchDetailForm(FlaskForm):
    launch_id = StringField()
    payloads = StringField()
    core_id = StringField()
    fairings_reused = SelectField(choices=[("True", "Yes"), ("False", "No")])
    fairings_recovery_attempts = SelectField(choices=[("True", "Yes"), ("False", "No")])
    fairings_recovered = SelectField(choices=[("True", "Yes"), ("False", "No")])
    failure_time = IntegerField()
    failure_altitude = IntegerField()
```

2.2 Parts Owned by Beyza Aydeniz

2.2.1 Rocket Details 1 Table

The Rocket Details 1 Table consists of 19 non-key fields about the engine information of the rocket such as a serial, engine type, engine layout, cost per launch, boosters and landing legs material. The Rocket Details 1 Table has a foreign key to the Rockets table. Rocket Details 1 table stores the engine information of rockets.

2.2.2 Engine Information Page

The view function parts for engine information in views.py:

```
@views.route('/rockets', methods=['GET', 'POST'])
def rockets():
    rocket_data = database.get_rockets()
    rocket_d1_data = database.get_rocket_d1()
```

```
    inexistent_rocket_d1 = []
```

```
    inexistent_rocket_d1_dict = []
```

```
    for rocket in rocket_data:
        for rocket_d1 in rocket_d1_data:
            if rocket["rocket_id"] == rocket_d1["rocket_id"]:
                present = True
                break
        if not present:
            inexistent_rocket_d1 += [rocket["rocket_id"]]
            inexistent_rocket_d1_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]
        present = False
```

```
    return render_template("rockets.html", rockets=rocket_data,
        rocket_d1s=rocket_d1_data, rocket_d2s=rocket_d2_data, rocket_images=rocket_image_data,
        inexistent_d1=inexistent_rocket_d1, inexistent_d2=inexistent_rocket_d2, inexistent_image=inexistent_rocket_image,
        inexistent_d1_dict=inexistent_rocket_d1_dict, inexistent_d2_dict=inexistent_rocket_d2_dict, inexistent_image_dict=inexistent_rocket_image_dict,
        formM=forms.RocketForm(), formD1=forms.RocketD1Form(), formD2=forms.RocketD2Form(), formI=forms.RocketImageForm())
```

SQL statement in database.py:

```
def get_rocket_d1():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            """SELECT * FROM rocket_details_1
            JOIN (SELECT rocket_id, name FROM rockets) AS rocket_names
            ON rocket_details_1.rocket_id = rocket_names.rocket_id"""
            ).fetchall()
```

2.2.3 Engine Information Deleting

The view function to delete engine information in views.py:

```
@views.route('/delete_rocket_detail_1', methods=['GET'])
def delete_rocket_detail_1():
    database.delete_rocket_d1(request.args.get("rocket_id"))
    return redirect(url_for("views.rockets"))
```

SQL statement for deleting engine information in database.py:

```
def delete_rocket_d1(rocket_id):
    with sqlite3.connect(db_location) as con:
        cursor = con.cursor()
        query = "DELETE FROM rocket_details_1 WHERE (rocket_id = ?)"
        cursor.execute(query, (rocket_id,))
        con.commit()
```

And the js function code for deleting engine information in index.js:

```
function delete_rocket_d1(rocket_id) {
    document.location = '/delete_rocket_detail_1?rocket_id=' + rocket_id
}
```

2.2.4 Engine Information Adding

Wtforms library is used for adding operation. The library consists of different fields for input and equivalent for <input> tag. The engine information form consisted of these items:

```
class RocketD1Form(FlaskForm):
    rocket_id = StringField()
    stages = IntegerField(validators=[validators.InputRequired()])
    boosters = IntegerField(validators=[validators.InputRequired()])
    cost_per_launch = IntegerField(validators=[validators.InputRequired()])
    landing_legs_number = IntegerField(validators=[validators.InputRequired()])
    landing_legs_material = StringField()
    engine_isp_sea_level = IntegerField(validators=[validators.InputRequired()])
    engine_isp_vacuum = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_sea_level_kN = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_sea_level_lbf = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_vacuum_kN = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_lbf = IntegerField(validators=[validators.InputRequired()])
    engine_number = IntegerField(validators=[validators.InputRequired()])
    engine_type = StringField()
    engine_version = StringField()
    engine_layout = StringField()
    engine_loss_max = IntegerField(validators=[validators.InputRequired()])
    engine_propellant_1 = StringField()
    engine_propellant_2 = StringField()
    engine_thrust_to_weight = DecimalField(validators=[validators.InputRequired()])
```

The view function to add engine information in views.py:

```
def add_rocket_detail_1():
    form = forms.RocketD1Form()
```

```
    if form.validate_on_submit():
        database.add_rocket_d1(request)
    return redirect(url_for("views.rockets"))
```

And the SQL statement for adding engine information in database.py:

```
def add_rocket_d1(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new rocket_d1
        rocket_d1_columns = cur.execute("PRAGMA table_info(rocket_details_1)").fetchall()

        new_rocket_d1 = []
        for column in rocket_d1_columns:
            if column["name"] in request.form.keys():
                new_rocket_d1 += [request.form[column["name"]]]
```

2.2.5 Engine Information Updating

Wtforms library is used for updating operation. The library consists of different fields for input and equivalent for <input> tag. The engine information form consisted of these items:

```
class RocketD1Form(FlaskForm):
    rocket_id = StringField()
    stages = IntegerField(validators=[validators.InputRequired()])
    boosters = IntegerField(validators=[validators.InputRequired()])
    cost_per_launch = IntegerField(validators=[validators.InputRequired()])
    landing_legs_number = IntegerField(validators=[validators.InputRequired()])
    landing_legs_material = StringField()
    engine_isp_sea_level = IntegerField(validators=[validators.InputRequired()])
    engine_isp_vacuum = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_sea_level_kN = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_sea_level_lbf = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_vacuum_kN = IntegerField(validators=[validators.InputRequired()])
    engine_thrust_lbf = IntegerField(validators=[validators.InputRequired()])
    engine_number = IntegerField(validators=[validators.InputRequired()])
    engine_type = StringField()
    engine_version = StringField()
    engine_layout = StringField()
    engine_loss_max = IntegerField(validators=[validators.InputRequired()])
    engine_propellant_1 = StringField()
    engine_propellant_2 = StringField()
    engine_thrust_to_weight = DecimalField(validators=[validators.InputRequired()])
```

The view function to update engine information in views.py:

```
@views.route('/update_rocket_detail_1', methods=['POST'])
def update_rocket_detail_1():
    form = forms.RocketD1Form()
    if form.validate_on_submit():
        database.update_rocket_d1(request)
```

```
    return redirect(url_for("views.rockets"))
```

And the SQL statement for updating engine information in database.py:

```
def update_rocket_d1(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update rocket_d1
        rocket_d1_columns = cur.execute("PRAGMA table_info(rocket_details_1)").fetchall()
        rocket_d1_columns_str = ",".join([column["name"] + "=?" for column in rocket_d1_columns if column["name"] != "rocket_id"])

        rocket_d1 = []
        for column in rocket_d1_columns:
            if column["name"] in request.form.keys():
                rocket_d1 += [request.form[column["name"]]]

        rocket_id = rocket_d1[0]
        rocket_d1 = rocket_d1[1:] + [rocket_id]

        cur.execute(f'UPDATE rocket_details_1 SET {rocket_d1_columns_str} WHERE rocket_id = ?', rocket_d1)
        con.commit()
```

2.2.6 Cores Table

The Cores Table consists of 8 non-key fields and its primary key is core ID. Cores table stores the information about cores. These informations are:

- Serial
- Status
- Reuse Count
- Block
- Real Time Locating System Attempts
- Real Time Locating System Landings
- Autonomous Spaceport Drone Ship Attempts
- Autonomous Spaceport Drone Ship Landings

2.2.7 Cores Page

The view function for core page in views.py:

```
@views.route('/cores', methods=['GET', 'POST'])
def cores():
    core_data = database.get_cores()
    return render_template("cores.html", cores=core_data, core_form = forms.CoresForm())
```

SQL statement in database.py:

```
def get_cores():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            "SELECT * FROM cores"
            ).fetchall()
```

```
    core_id = -1
    for core in database.get_cores():
        current_id = core["core_id"]
        if current_id.isdigit():
            if int(current_id) > int(core_id):
                core_id = current_id
    app.config["core_id"] = int(core_id) + 1
```

To show datas in table

```
<header>
    <button type="button" class="btn btn-warning btn" data-toggle="modal"
   data-target="#add-core-modal" style="position: fixed; bottom: 5em; right
   : 2.5em;">
        New Core
    </button>
    <button type="button" class="btn btn-warning" data-toggle="modal" data-
   target="#filter-core-modal" style="position: fixed; bottom: 2.5em; right
   : 2.5em;">
        Filter Core
    </button>
<table class = "table table-hover table-sm">
    <tr class = "table-active" style="text-align: center;">
        <th>Serial</th>
        <th>Status</th>
        <th>Reuse Count</th>
        <th>Block</th>
        <th>Real Time Locating System Attempts</th>
        <th>Real Time Locating System Landings</th>
        <th>Autonomous Spaceport Drone Ship Attempts</th>
        <th>Autonomous Spaceport Drone Ship Landings</th>
        <th colspan="2">Edit</th>
    </tr>
    {% for core in cores %}
    <tr class style="text-align: center;">
        <td>
            {{core["serial"]}}
        </td>
        <td>
            {{core["status"]}}
        </td>
        <td>
            {{core["reuse_count"]}}
        </td>
        <td>
            {{core["block"]}}
        </td>
        <td>
            {{core["rtls_attempts"]}}
        </td>
        <td>
            {{core["rtls_landings"]}}
```

```
        </td>
        <td>
            {{core["asds_attempts"]}}
        </td>
        <td>
            {{core["asds_landings"]}}
        </td>
        <td>
            <button type="button" class="btn btn-danger btn-sm" data-toggle
    ="modal" onclick="delete_core('{{ core.core_id }}')">
                Delete
            </button>
        </td>
        <td>
            <button type="button" class="btn btn-warning btn-sm" data-
    toggle="modal" data-target="#update-core-modal-{{ core.core_id }}">
              Update
            </button>
        </td>
    </tr>
    {% endfor %}
</table>
```

2.2.8 Core Deleting

The view function to delete core in views.py:

```
@views.route('/delete_core', methods=['GET'])
@login_required
def delete_core():
    if current_user.is_admin:
        database.delete_core(request.args.get("core_id"))
    return redirect(url_for("views.cores"))
```

Delete button for relevant line in table

```
        <td>
            <button type="button" class="btn btn-danger btn-sm" data-toggle
    ="modal" onclick="delete_core('{{ core.core_id }}')">
                Delete
            </button>
        </td>
```

SQL statement for deleting core in database.py:

```
def delete_core(core_id):
    with sqlite3.connect(db_location) as con:
        cursor = con.cursor()
        cursor.execute("PRAGMA foreign_keys=ON")
        query = "DELETE FROM cores WHERE (core_id = ?)"
        cursor.execute(query, (core_id,))
        con.commit()
```

And the js function code for deleting core in index.js:

```
function delete_core(core_id) {
    document.location = '/delete_core?core_id=' + core_id
}
```

2.2.9 Core Adding

Wtforms library is used for adding operation. The library consists of different fields for input and equivalent for <input> tag. The cores form consisted of these items:

```
class CoresForm(FlaskForm):
    core_id = StringField()
    serial = StringField()
    status = StringField()
    reuse_count = IntegerField()
    block = IntegerField()
    rtls_attempts = IntegerField()
    rtls_landings = IntegerField()
    asds_attempts = IntegerField()
    asds_landings = IntegerField()
```

The view function to add core in views.py:

```
@views.route("/add_core", methods=["POST"])
@login_required
def add_core():
    if current_user.is_admin:
        database.add_core(request)
    return redirect(url_for("views.cores"))
```

And the SQL statement for adding core in database.py:

```
def add_core(request):
  with sqlite3.connect(db_location) as con:
    con.row_factory = sqlite3.Row
    cur = con.cursor()

    core_columns = cur.execute("PRAGMA table_info(cores)").fetchall()

    new_core = [str(current_app.config["core_id"])]
    current_app.config["core_id"] += 1
    for column in core_columns:
      if column["name"] in request.form.keys():
        new_core += [request.form[column["name"]]]

    cur.execute(f'INSERT INTO cores VALUES ({",".join("?" * len(
    core_columns))})', new_core)
    con.commit()
```

Core adding form in cores.html:

```
<div class="modal-body"><form action="/add_core" method="POST" style="
    display: grid;">
  {{ core_form.csrf_token }}
  <div class="field">
    <label for="serial" class="form-label detail-field">Serial:</label>
    <div class="control detail-field">
      {{ core_form.serial(class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="status" class="form-label detail-field">Status:</label>
    <div class="control detail-field">
      {{ core_form.status(class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="reuse_count" class="form-label detail-field">Reuse Count:</
    label>
```

```
    <div class="control detail-field">
      {{ core_form.reuse_count(class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="block" class="form-label detail-field">Block:</label>
    <div class="control detail-field">
      {{ core_form.block(class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="rtls_attempts" class="form-label detail-field">Real Time Locating System Attempts:</label>
    <div class="control detail-field">
      {{ core_form.rtls_attempts(class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="rtls_landings" class="form-label detail-field">Real Time Locating System Landings:</label>
    <div class="control detail-field">
      {{ core_form.rtls_landings(class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="asds_attempts" class="form-label detail-field">Autonomous Spaceport Drone Ship Attempts:</label>
    <div class="control detail-field">
      {{ core_form.asds_attempts(class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="asds_landings" class="form-label detail-field">Autonomous Spaceport Drone Ship Landings:</label>
    <div class="control detail-field">
      {{ core_form.asds_landings(class="form-control-sm")}}
    </div>
  </div>
  <button type=submit class="btn btn-warning">Add</button>
  </form>
</div>
```

”New Core” button in the header

```
    <button type="button" class="btn btn-warning btn" data-toggle="modal" data-target="#add-core-modal" style="position: fixed; bottom: 5em; right: 2.5em;">
        New Core
    </button>
```

2.2.10 Core Updating

Wtforms library is used for updating operation. The library consists of different fields for input and equivalent for <input> tag. The cores form consisted of these items:

```
class CoresForm(FlaskForm):
    core_id = StringField()
    serial = StringField()
    status = StringField()
    reuse_count = IntegerField()
    block = IntegerField()
```

```
    rtls_attempts = IntegerField()
    rtls_landings = IntegerField()
    asds_attempts = IntegerField()
    asds_landings = IntegerField()
```

The view function to update core in views.py:

```
@views.route("/update_core", methods=["POST"])
@login_required
def update_core():
    if current_user.is_admin:
        database.update_core(request)
    return redirect(url_for("views.cores"))
```

And the SQL statement for updating core in database.py:

```
def update_core(request):
 with sqlite3.connect(db_location) as con:
    con.row_factory = sqlite3.Row
    cur = con.cursor()

    core_columns = cur.execute("PRAGMA table_info(cores)").fetchall()
    core_columns_str = ",".join([column["name"] + "=?" for column in
   core_columns if column["name"] != "core_id"])

    core = []
    for column in core_columns:
      if column["name"] in request.form.keys():
        core += [request.form[column["name"]]]

    core_id = core[0]
    core = core[1:] + [core_id]

    cur.execute(f'UPDATE cores SET {core_columns_str} WHERE core_id = ?',
   core)
    con.commit()
```

Core updating form in cores.html:

```
<div class="modal-body"><form action="/update_core" method="POST" style="
   display: grid;">
  {{ core_form.csrf_token }}
  <div class="field" style="display: none">
    <div class="control">
      {{ core_form.core_id(value=core.core_id, class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="serial" class="form-label detail-field">Serial:</label>
    <div class="control detail-field">
      {{ core_form.serial(value=core.serial,class="form-control-sm")}}
      </div>
  </div>
  <div class="field">
    <label for="status" class="form-label detail-field">Status:</label>
    <div class="control detail-field">
      {{ core_form.status(value=core.status,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="reuse_count" class="form-label detail-field">Reuse Count:</
   label>
    <div class="control detail-field">
```

```
      {{ core_form.reuse_count(value=core.reuse_count,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="block" class="form-label detail-field">Block:</label>
    <div class="control detail-field">
      {{ core_form.block(value=core.block,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="rtls_attempts" class="form-label detail-field">Real Time Locating System Attempts:</label>
    <div class="control detail-field">
      {{ core_form.rtls_attempts(value=core.rtls_attempts,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="rtls_landings" class="form-label detail-field">Real Time Locating System Landings:</label>
    <div class="control detail-field">
      {{ core_form.rtls_landings(value=core.rtls_landings,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="asds_attempts" class="form-label detail-field">Autonomous Spaceport Drone Ship Attempts:</label>
    <div class="control detail-field">
      {{ core_form.asds_attempts(value=core.asds_attempts,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="asds_landings" class="form-label detail-field">Autonomous Spaceport Drone Ship Landings:</label>
    <div class="control detail-field">
      {{ core_form.asds_landings(value=core.asds_landings,class="form-control-sm")}}
    </div>
  </div>
  <button type=submit class="btn btn-warning">Update</button>
  </form>
</div>
```

Update button for relevant line in table

```
        <td>
            <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-target="#update-core-modal-{{ core.core_id }}">
              Update
            </button>
        </td>
```

2.2.11 Core Filtering

Wtforms library is used for filtering operation. The library consists of different fields for input and equivalent for <input> tag. The cores form consisted of these items:

```
class CoresForm(FlaskForm):
```

```
    core_id = StringField()
    serial = StringField()
    status = StringField()
    reuse_count = IntegerField()
    block = IntegerField()
    rtls_attempts = IntegerField()
    rtls_landings = IntegerField()
    asds_attempts = IntegerField()
    asds_landings = IntegerField()
```

The view function to filter core in views.py:

```
@views.route('/filter_core', methods=['GET', 'POST'])
def filter_core():
    filter_core_data = database.filter_core(request)
    return render_template("cores.html", cores=filter_core_data, core_form
   = forms.CoresForm())
```

And the SQL statement for filtering core in database.py:

```
def filter_core(request):
  with sqlite3.connect(db_location) as con:
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    print("filter_core executed")
    query = "SELECT * FROM cores"
    params = []
    print(request.form.get('fserial'))
    if request.form.get('fserial'):
      if not params:
        query += " WHERE"
      user_serial = request.form['fserial']
      query += " serial LIKE ?"
      param_serial = "%" + user_serial + "%"
      params.append(param_serial)
      if request.form.get('fstatus'):
        if not params:
          query += " WHERE"
        else:
          query += " AND"
        user_status = request.form['fstatus']
        query += " status LIKE ?"
        param_status = "%" + user_status + "%"
        params.append(param_status)
      if request.form.get('freuse_count') != "":
        if not params:
          query += " WHERE"
        else:
          query += " AND"
        user_reuse_count = request.form['freuse_count']
        query += " reuse_count LIKE ?"
        param_reuse_count = "%" + user_reuse_count + "%"
        params.append(param_reuse_count)

      if request.form.get('fblock'):
        if not params:
          query += " WHERE"
        else:
          query += " AND"
        user_block = request.form['fblock']
        query += " block LIKE ?"
        param_block = "%" + user_block + "%"
```

```
        params.append(param_block)

      if request.form.get('frtls_attempts'):
        if not params:
          query += " WHERE"
        else:
          query += " AND"
        user_rtls_attempts = request.form['frtls_attempts']
        query += " rtls_attempts LIKE ?"
        param_rtls_attempts = "%" + user_rtls_attempts + "%"
        params.append(param_rtls_attempts)

      if request.form.get('frtls_landings'):
        if not params:
          query += " WHERE"
        else:
          query += " AND"
        user_rtls_landings = request.form['frtls_landings']
        query += " rtls_landings LIKE ?"
        param_rtls_landings = "%" + user_rtls_landings + "%"
        params.append(param_rtls_landings)

      if request.form.get('fasds_attempts'):
        if not params:
          query += " WHERE"
        else:
          query += " AND"
        user_asds_attempts = request.form['fasds_attempts']
        query += " asds_attempts LIKE ?"
        param_asds_attempts = "%" + user_asds_attempts + "%"
        params.append(param_asds_attempts)
      if request.form.get('fasds_landings'):
        if not params:
          query += " WHERE"
        else:
          query += " AND"
        user_asds_landings = request.form['fasds_landings']
        query += " asds_landings LIKE ?"
        param_asds_landings = "%" + user_asds_landings + "%"
        params.append(param_asds_landings)

    print(query)
    print(tuple(params))
    filter = cur.execute(query, tuple(params)).fetchall()
    return filter
```

Core filtering form in cores.html:

```
<div class="modal-body"><form action="/update_core" method="POST" style="
    display: grid;">
  {{ core_form.csrf_token }}
  <div class="field" style="display: none">
    <div class="control">
      {{ core_form.core_id(value=core.core_id, class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="serial" class="form-label detail-field">Serial:</label>
    <div class="control detail-field">
      {{ core_form.serial(value=core.serial,class="form-control-sm")}}
      </div>
  </div>
```

```
  <div class="field">
    <label for="status" class="form-label detail-field">Status:</label>
    <div class="control detail-field">
      {{ core_form.status(value=core.status,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="reuse_count" class="form-label detail-field">Reuse Count:</label>
    <div class="control detail-field">
      {{ core_form.reuse_count(value=core.reuse_count,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="block" class="form-label detail-field">Block:</label>
    <div class="control detail-field">
      {{ core_form.block(value=core.block,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="rtls_attempts" class="form-label detail-field">Real Time Locating System Attempts:</label>
    <div class="control detail-field">
      {{ core_form.rtls_attempts(value=core.rtls_attempts,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="rtls_landings" class="form-label detail-field">Real Time Locating System Landings:</label>
    <div class="control detail-field">
      {{ core_form.rtls_landings(value=core.rtls_landings,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="asds_attempts" class="form-label detail-field">Autonomous Spaceport Drone Ship Attempts:</label>
    <div class="control detail-field">
      {{ core_form.asds_attempts(value=core.asds_attempts,class="form-control-sm")}}
    </div>
  </div>
  <div class="field">
    <label for="asds_landings" class="form-label detail-field">Autonomous Spaceport Drone Ship Landings:</label>
    <div class="control detail-field">
      {{ core_form.asds_landings(value=core.asds_landings,class="form-control-sm")}}
    </div>
  </div>
  <button type=submit class="btn btn-warning">Update</button>
  </form>
</div>
```

Filter button in the header

```
    <button type="button" class="btn btn-warning" data-toggle="modal" data-target="#filter-core-modal" style="position: fixed; bottom: 2.5em; right: 2.5em;">
        Filter Core
```

2.3 Parts Owned by Abdullah Asım Emül

2.3.1 Code Structure

Each page and functionality of our app is provided through a similar pattern of actions:

1. The user sends a request
2. The request is directed using a `views.py` configuration
3. DB operations are performed as specified in `database.py`
4. Any related WTForms are read from `forms.py`
5. The resultant page is created by Jinja using a template in the `templates/` folder and the data sent to it in `views.py` via a `render_template()` call

2.3.2 User Management

Login If a user wishes to login, the form presented is described as follows:

`forms.py`

```
class LoginForm(FlaskForm):
    username = StringField("Username", validators=[validators.DataRequired()])
    password = PasswordField("Password", validators=[validators.DataRequired()])
```

`login.html`

```
        <form action="/login" method="POST">
            {{ form.csrf_token }}
            <div class="field" style="margin: 90px 0px 0px 80px;">
                <label for="username" class="form-label" style="color:white;">Username:</label>
                <div class="control">
                    {{ form.username(required=True, class="form-control-lg")}}
                </div>
            </div>
            <br>
            <div class="field" style="margin: 0px 0px 0px 80px;">
                <label for="password" class="form-label" style="color:white;">Password:</label>
                <div class="control">
                    {{ form.password(required=True, class="form-control-lg")}}
                </div>
            </div>

            <button type=submit class="btn btn-warning" style="margin: 20px 20px 20px 90px;">Log in</button>
        </form>
```

Once this form is sent via a POST request to our backend, it is processed as so:

`views.py`

```
def login():
    form = forms.LoginForm()
    if form.validate_on_submit():
        username = form.data["username"]
```

```
        user = users.get_user(username)

        if user is not None:
            password = form.data["password"]
            if passlib.hash.pbkdf2_sha256.verify(password, user.password):
                login_user(user)
                flash("Welcome to SpaceXhibit!")

                return redirect(url_for("views.home"))
            else:
                flash("Wrong password.")
        else:
            flash("You're not on the guest list. Why don't you sign up?")

    return render_template("login.html", form=form)
```

If the credentials are correct, the user is sent a "welcome" message, logged in via Flask-Login, and redirected to the home page. If the username does not exist (which is checked as shown below), the user is prompted to sign up while if the password is wrong, this information is relayed to the user.

users.py

```
def get_user(username):
    user_db = sqlite3.connect(user_db_location)
    user_db.row_factory = sqlite3.Row
    user_data = user_db.cursor().execute("SELECT * FROM users WHERE
   username = ?", [username]).fetchall()

    if len(user_data) == 1:
        return User(user_data[0]["username"], user_data[0]["password"],
   user_data[0]["type"])
    else:
        return None
```

Signup Users are presented with the following form when they request the signup page:

forms.py

```
class SignupForm(FlaskForm):
    username = StringField(validators=[validators.DataRequired()])
    password = PasswordField(validators=[validators.DataRequired()])
    superfan = RadioField(choices=[("True", "Yes"), ("False", "No")],
   validators=[validators.DataRequired()])
```

signup.html

```
    <form action="/signup" method="POST">
        {{ form.csrf_token }}
        <div class="field" style="margin: 90px 0px 0px 80px;">
            <label for="username" class="form-label label-color" style="
   color:white;">Username:</label>
            <div class="control">
                {{ form.username(required=True, class="form-control-lg")}}
            </div>
        </div>
        <br>
        <div class="field" style="margin: 0px 0px 0px 80px;">
            <label for="password" class="form-label" style="color:white;">
   Password:</label>
            <div class="control">
                {{ form.password(required=True, class="form-control-lg")}}
            </div>
```

```
        </div>
        <br>
        <div class="field" style="margin: 0px 0px 0px 80px;">
            <label for="superfan" class="form-label" style="color:white;">Are you a superfan of SpaceX?</label>
            <div class="control" style="color:white;">
                {{ form.superfan(required=True, class="form-control-lg list-style-none")}}
            </div>
        </div>
        <div class="field" style="margin: 0px 0px 0px 80px;">
            <div class="control" style="color:white;">
                <input type="checkbox" name="wants_admin">
                <label for="wants_admin" class="form-label" style="color:white;">I would like to be an admin.</label>
            </div>
        </div>

        <button type=submit class="btn btn-warning" style="margin: 10px 20px 20px 80px; color:white;">Sign up</button>
    </form>
```

Once the user sends all relevant information, it is added to the database. The password is hashed. If the user selected "True" for being a superfan and they checked the box to be considered for adminhood, their user type will be set as admin. Otherwise, it will be regular.

`users.py`

```
def add_user(username, password, type):
    user_db = sqlite3.connect(user_db_location)
    user_db.row_factory = sqlite3.Row

    user_db.cursor().execute(
        'INSERT INTO users (username, password, type) VALUES (?, ?, ?)',
        (username, password, type))
    user_db.commit()
```

After this, the user is taken to the login page as follows:

`views.py`

```
def signup():
    form = forms.SignupForm()
    if form.validate_on_submit():
        username = form.data["username"]
        user = users.get_user(username)

        if user is None:
            password = form.data["password"]
            hashed_password = \
                passlib.hash.pbkdf2_sha256.hash(password)

            user_type = "regular"
            deserves_adminhood = True if form.data["superfan"] == "True" else False
            if deserves_adminhood and "wants_admin" in request.form.keys():
                user_type = "admin"

            users.add_user(username, hashed_password, user_type)

            flash("You're all set, you can log in.")
            return redirect(url_for("views.login"))
        else:
            flash("That username is taken, please choose another.")
```

```

    return render_template("signup.html", form=form)
```

2.3.3 Rockets (Main)

Rockets (Main) Viewing Page Users perform all actions related to rockets by first landing on the rocket viewing page. This page is served via the following function:

views.py

```
@views.route('/rockets', methods=['GET', 'POST'])
def rockets():
    rocket_data = database.get_rockets()
    rocket_d1_data = database.get_rocket_d1()
    rocket_d2_data = database.get_rocket_d2()
    rocket_image_data = database.get_rocket_image()

    inexistent_rocket_d1 = []
    inexistent_rocket_d2 = []
    inexistent_rocket_image = []
    inexistent_rocket_d1_dict = []
    inexistent_rocket_d2_dict = []
    inexistent_rocket_image_dict = []

    present = False
    for rocket in rocket_data:
        for rocket_d1 in rocket_d1_data:
            if rocket["rocket_id"] == rocket_d1["rocket_id"]:
                present = True
                break
        if not present:
            inexistent_rocket_d1 += [rocket["rocket_id"]]
            inexistent_rocket_d1_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]
        present = False

        for rocket_d2 in rocket_d2_data:
            if rocket["rocket_id"] == rocket_d2["rocket_id"]:
                present = True
                break
        if not present:
            inexistent_rocket_d2 += [rocket["rocket_id"]]
            inexistent_rocket_d2_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]
        present = False

        for rocket_image in rocket_image_data:
            if rocket["rocket_id"] == rocket_image["rocket_id"]:
                present = True
                break
        if not present:
            inexistent_rocket_image += [rocket["rocket_id"]]
            inexistent_rocket_image_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]
        present = False

    return render_template("rockets.html", rockets=rocket_data,
        rocket_d1s=rocket_d1_data, rocket_d2s=rocket_d2_data, rocket_images=rocket_image_data,
        inexistent_d1=inexistent_rocket_d1, inexistent_d2=inexistent_rocket_d2, inexistent_image=inexistent_rocket_image,
```

```
        inexistent_d1_dict=inexistent_rocket_d1_dict, inexistent_d2_dict=inexistent_rocket_d2_dict, inexistent_image_dict=inexistent_rocket_image_dict,
        formM=forms.RocketForm(), formD1=forms.RocketD1Form(), formD2=forms.RocketD2Form(), formI=forms.RocketImageForm())
```

This code involves checking which rockets have details and which do not, along with name information for those that do not. There is a list of rockets which do not have certain details so that to check when generating the template of whether to use a "view" button or an "add" button.

The database interaction to get rocket information is as follows:

`database.py`

```
def get_rockets():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            "SELECT * FROM rockets"
            ).fetchall()
```

The data is then returned in a table generated using the following Jinja template:

`rockets.html`

```
<table class = "table table-hover table-sm">
    <thead>
        <tr class = "table-active">
            <th> </th>
            <th>Name</th>
            <th>Type</th>
            <th>Status</th>
            <th>Country of Origin</th>
            <th>Company</th>
            <th colspan="2">Details</th>
            <th colspan="2">Editing</th>
        </tr>
    </thead>
    <tbody>
        {% for rocket in rockets %}
        <tr class>
            <td>
              {% if rocket["rocket_id"] in inexistent_image %}
                <button type="button" class="btn btn-sm" data-toggle="modal" data-target="#add-rocket-image-modal-{{ rocket.rocket_id }}">

                </button>
              {% else %}
                <button type="button" class="btn btn-sm" data-toggle="modal" data-target="#rocket-image-modal-{{ rocket.rocket_id }}">

                </button>
              {% endif %}
            </td>
            <td> {{rocket["name"]}} </td>
            <td> {{rocket["type"]}} </td>
            <td> {{ "Active" if rocket["active"] == "True" else "Decommissioned" }} </td>
            <td> {{rocket["country"]}} </td>
            <td> {{rocket["company"]}}  </td>

            <td>
              {% if rocket["rocket_id"] in inexistent_d1 %}
```

```
                <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-target="#add-rocket-details-1-modal-{{ rocket.rocket_id }}">
                  Add Engine Info
                </button>
              {% else %}
                <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-target="#rocket-details-1-modal-{{ rocket.rocket_id }}">
                  Engine Info
                </button>
              {% endif %}
            </td>
            <td>
              {% if rocket["rocket_id"] in inexistent_d2 %}
                <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-target="#add-rocket-details-2-modal-{{ rocket.rocket_id }}">
                  Add Measurements
                </button>
              {% else %}
                <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-target="#rocket-details-2-modal-{{ rocket.rocket_id }}">
                  Measurements
                </button>
              {% endif %}
            </td>
            <td>
                <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-target="#update-rocket-modal-{{ rocket.rocket_id }}">
                  Update
                </button>
            </td>
            <td>
                {% if current_user.is_admin %}
                  <button type="button" class="btn btn-danger btn-sm" data-toggle="modal"
                  onclick="delete_rocket('{{ rocket.rocket_id }}')">
                    Delete
                  </button>
                {% endif %}
            </td>


        </tr>
        {% endfor %}
    </tbody>
</table>
```

Rockets (Main) Adding Modal When a user wishes to add a new rocket into the system, they click the "New Rocket" button defined as follows:

rockets.html

```
<button type="button" class="btn btn-warning" data-toggle="modal" data-target="#add-rocket-modal" style="position: fixed; bottom: 5em; right: 2.5em;">
  New Rocket
</button>
```

This interaction opens a modal containing a form defined as so:

rockets.html

```

<div class="modal fade" id="add-rocket-modal" tabindex="-1" role="dialog" aria-labelledby="add-rocket-modal-label" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="rocket-modal-label"> Add Rocket </h5>
      </div>
      <div class="modal-body"><form action="/add_rocket" method="POST" style="display: grid;">
        {{ formM.csrf_token }}
        <div class="field">
            <label for="name" class="form-label detail-field">Name:</label>
            <div class="control detail-field">
                {{ formM.name(class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="type" class="form-label detail-field">Type:</label>
            <div class="control detail-field">
                {{ formM.type(class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="active" class="form-label detail-field">Active:</label>
            <div class="control detail-field">
                {{ formM.active(class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="country" class="form-label detail-field">Country:</label>
            <div class="control detail-field">
                {{ formM.country(class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="company" class="form-label detail-field">Company:</label>
            <div class="control detail-field">
                {{ formM.company(class="form-control-sm")}}
            </div>
        </div>

        <button type=submit class="btn btn-warning">Add</button>
      </form>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
      </div>
    </div>
  </div>
</div>
```

forms.py

```
class RocketForm(FlaskForm):
```

```
    rocket_id = StringField()
    name = StringField()
    type = StringField()
    active = SelectField(choices=[("True", "Active"), ("False", "Decommissioned")])
    country = StringField()
    company = StringField()
```

Once the user fills in the form and sends it, the request is handled by the `add_rocket()` function:

`views.py`

```
def add_rocket():
    form = forms.RocketForm()
    if form.validate_on_submit():
        database.add_rocket(request)
    return redirect(url_for("views.rockets"))
```

Here, the form is validated and sent to the database as follows:

`database.py`

```
def add_rocket(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new rocket
        rocket_columns = cur.execute("PRAGMA table_info(rockets)").fetchall()

        new_rocket = [str(current_app.config["rocket_id"])]  # Get next ID
        current_app.config["rocket_id"] += 1
        for column in rocket_columns:
            if column["name"] in request.form.keys():
                new_rocket += [request.form[column["name"]]]

        cur.execute(f'INSERT INTO rockets VALUES ({",".join("?" * len(rocket_columns))})', new_rocket)
        con.commit()
```

Here, it is important to note that due to our data not having integer IDs for data already present in it (it has strings instead), we had to generate IDs ourselves sequentially for any new additions we were to make.

After these steps, the user is redirected to the rocket viewing page where they can see their addition.

Rockets (Main) Editing Modal When a user wishes to edit a rocket, they open a modal created with the same WTForm as rocket addition but with the rocket ID already entered in the template using Jinja.

`rockets.html`

```

{% for rocket in rockets %}

<div class="modal fade" id="update-rocket-modal-{{ rocket.rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="update-rocket-modal-label-{{ rocket.rocket_id }}" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
```

```
8           <h5 class="modal-title" id="rocket-modal-label-{{ rocket.
    rocket_id }}"> Update {{ rocket.name }} </h5>
9       </div>
10      <div class="modal-body"><form action="/update_rocket" method="POST"
     style="display: grid;">
11          {{ formM.csrf_token }}
12          <div class="field" style="display: none">
13              <div class="control">
14                  {{ formM.rocket_id(value=rocket.rocket_id, class="form-
    control-sm")}}
15              </div>
16          </div>
17          <div class="field">
18              <label for="name" class="form-label detail-field">Name:</
    label>
19              <div class="control detail-field">
20                  {{ formM.name(value=rocket.name,class="form-control-sm")
    }}
21              </div>
22          </div>
23          <div class="field">
24              <label for="type" class="form-label detail-field">Type:</
    label>
25              <div class="control detail-field">
26                  {{ formM.type(value=rocket.type,class="form-control-sm")
    }}
27              </div>
28          </div>
29          <div class="field">
30              <label for="active" class="form-label detail-field">Active:</
    label>
31              <div class="control detail-field">
32                  <!-- {{ formM.active(value=rocket.active,class="form-
    control-sm")}} -->
33                  <select class="form-control-sm" id="active" name="active"
     value="True">
34                    {% if rocket.active == "True" %}
35                      <option value="True" selected>Active</option>
36                      <option value="False">Decommissioned</option>
37                    {% else %}
38                    <option value="True">Active</option>
39                    <option value="False" selected>Decommissioned</option>
40                    {% endif %}
41                  </select>
42              </div>
43          </div>
44          <div class="field">
45              <label for="country" class="form-label detail-field">Country:
    </label>
46              <div class="control detail-field">
47                  {{ formM.country(value=rocket.country,class="form-control
    -sm")}}
48              </div>
49          </div>
50          <div class="field">
51              <label for="company" class="form-label detail-field">Company:
    </label>
52              <div class="control detail-field">
53                  {{ formM.company(value=rocket.company,class="form-control
    -sm")}}
```

```
                </div>
            </div>

            <button type=submit class="btn btn-warning">Update</button>
          </form>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
          </div>
        </div>
      </div>
    </div>

{% endfor %}
```

When the user submits this form, it is handled by the following code:

`views.py`

```
@views.route("/update_rocket", methods=["POST"])
def update_rocket():
    form = forms.RocketForm()
    if form.validate_on_submit():
        database.update_rocket(request)
    return redirect(url_for("views.rockets"))
```

The database operations are performed as follows:

`database.py`

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update rocket
        rocket_columns = cur.execute("PRAGMA table_info(rockets)").fetchall()
        rocket_columns_str = ",".join([column["name"] + "=?" for column in rocket_columns if column["name"] != "rocket_id"])

        rocket = []
        for column in rocket_columns:
            if column["name"] in request.form.keys():
                rocket += [request.form[column["name"]]]

        rocket_id = rocket[0]
        rocket = rocket[1:] + [rocket_id]

        cur.execute(f'UPDATE rockets SET {rocket_columns_str} WHERE rocket_id = ?', rocket)
        con.commit()
def update_rocket_d1(request):
    with sqlite3.connect(db_location) as con:
```

After these steps, the user is redirected to the rocket viewing page where they can see their edit.

Rocket (Main) Deletion Users can only delete rockets if they are logged in with an admin account. The delete button is only shown in this case while the backend checks this condition in the following function:

`views.py`

```
@views.route('/delete_rocket', methods=['GET'])
```

```
@login_required
def delete_rocket():
    if current_user.is_admin:
        database.delete_rocket(request.args.get("rocket_id"))
    else:
        flash("Please do not poke around the exhibit.")
    return redirect(url_for("views.rockets"))
```

When an admin user clicks the delete button of a rocket, they send a request with the rocket's ID. This is then processed by the following function:

database.py

```
        cursor = con.cursor()
        cursor.execute("PRAGMA foreign_keys=ON")  # This allows for cascading to details
        query = "DELETE FROM rockets WHERE (rocket_id = ?)"
        cursor.execute(query, (rocket_id,))
        con.commit()
def delete_rocket_d1(rocket_id):
    with sqlite3.connect(db_location) as con:
```

After these steps, the user is redirected to the rocket viewing page where they can see their deletion.

2.3.4 Rocket Measurements

Rocket Measurements Viewing Page When a user wishes to view a rocket's measurements, they can click on the "Measurements" button of the respective rocket which is generated as follows:

rockets.html

```

{% for rocket_d2 in rocket_d2s %}
<div class="modal fade" id="update-rocket-details-2-modal-{{ rocket_d2.rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="update-rocket-details-2-modal-label-{{ rocket_d2.rocket_id }}" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="update-rocket-details-2-modal-label-{{ rocket_d2.rocket_id }}"> Update {{rocket_d2.name}} Measurements </h5>
        </div>
        <div class="modal-body">
          <form action="/update_rocket_detail_2" method="POST" style="display: grid;">
          {{ formD2.csrf_token }}
          <div class="field" style="display: none">
              <div class="control">
                  {{ formD2.rocket_id(value=rocket_d2.rocket_id, class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="height_mt" class="form-label detail-field">Height (m):</label>
              <div class="control detail-field">
                  {{ formD2.height_mt(value=rocket_d2.height_mt, class="form-control-sm")}}
              </div>
```

```
        </div>
        <div class="field">
            <label for="height_ft" class="form-label detail-field">Height (ft):</label>
            <div class="control detail-field">
                {{ formD2.height_ft(value=rocket_d2.height_ft, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="diameter_mt" class="form-label detail-field">Diameter (m):</label>
            <div class="control detail-field">
                {{ formD2.diameter_mt(value=rocket_d2.diameter_mt, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="diameter_ft" class="form-label detail-field">Diameter (ft):</label>
            <div class="control detail-field">
                {{ formD2.diameter_ft(value=rocket_d2.diameter_ft, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="mass_kg" class="form-label detail-field">Mass (kg):</label>
            <div class="control detail-field">
                {{ formD2.mass_kg(value=rocket_d2.mass_kg, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="mass_lb" class="form-label detail-field">Mass (lb):</label>
            <div class="control detail-field">
                {{ formD2.mass_lb(value=rocket_d2.mass_lb, class="form-control-sm")}}
            </div>
        </div>

        <button type=submit class="btn btn-warning">Update</button>
      </form>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
      </div>
    </div>
  </div>
</div>
{% endfor %}
```

This information is always included in the page served for rockets in general. The data is retrieved from the database as follows:

database.py

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
```

```
            """SELECT * FROM rocket_details_2
            JOIN (SELECT rocket_id, name FROM rockets) AS rocket_names
            ON rocket_details_2.rocket_id = rocket_names.rocket_id"""
            ).fetchall()
def get_rocket_image():
    with sqlite3.connect(db_location) as con:
```

The `rocket_details_2` table is joined with the `rockets` table to allow for the name of the relevant rocket to be readily available to be passed to Jinja.

Rocket Measurements Adding Modal When a user wishes to add measurement information to a rocket via the "Add Measurements" button, they are presented with modal defined as follows:

`rockets.html`

```

{% for rocket in inexistent_d2_dict %}
<div class="modal fade" id="add-rocket-details-2-modal-{{ rocket.rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="add-rocket-details-2-modal-label-{{ rocket.rocket_id }}" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="rocket-details-2-modal-label-{{ rocket.rocket_id }}"> Add {{rocket.name}} Measurements </h5>
        </div>
        <div class="modal-body">
          <form action="/add_rocket_detail_2" method="POST" style="display: grid;">
          {{ formD2.csrf_token }}
          <div class="field" style="display: none">
              <div class="control">
                  {{ formD2.rocket_id(value=rocket.rocket_id, class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="height_mt" class="form-label detail-field">Height (m):</label>
              <div class="control detail-field">
                  {{ formD2.height_mt(class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="height_ft" class="form-label detail-field">Height (ft):</label>
              <div class="control detail-field">
                  {{ formD2.height_ft(class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="diameter_mt" class="form-label detail-field">Diameter (m):</label>
              <div class="control detail-field">
                  {{ formD2.diameter_mt(class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="diameter_ft" class="form-label detail-field">Diameter (ft):</label>
```

```
                <div class="control detail-field">
                    {{ formD2.diameter_ft(class="form-control-sm")}}
                </div>
            </div>
            <div class="field">
                <label for="mass_kg" class="form-label detail-field">Mass (kg):</label>
                <div class="control detail-field">
                    {{ formD2.mass_kg(class="form-control-sm")}}
                </div>
            </div>
            <div class="field">
                <label for="mass_lb" class="form-label detail-field">Mass (lb):</label>
                <div class="control detail-field">
                    {{ formD2.mass_lb(class="form-control-sm")}}
                </div>
            </div>

            <button type=submit class="btn btn-warning">Add</button>
        </form>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
        </div>
      </div>
    </div>
  </div>
{% endfor %}
```

This modal's form is defined as follows:

`forms.py`

```
class RocketD2Form(FlaskForm):
    rocket_id = StringField()
    height_mt = DecimalField(validators=[validators.InputRequired()])
    height_ft = DecimalField(validators=[validators.InputRequired()])
    diameter_mt = DecimalField(validators=[validators.InputRequired()])
    diameter_ft = DecimalField(validators=[validators.InputRequired()])
    mass_kg = IntegerField(validators=[validators.InputRequired()])
    mass_lb = IntegerField(validators=[validators.InputRequired()])
```

Once the user submits the form, if the data is valid, it is sent to the database handler as follows:

`views.py`

```
@views.route('/add_rocket_detail_2', methods=['POST'])
def add_rocket_detail_2():
    form = forms.RocketD2Form()
    if form.validate_on_submit():
        database.add_rocket_d2(request)
    return redirect(url_for("views.rockets"))
```

The database connection is then handled as follows:

`database.py`

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new rocket_d2
        rocket_d2_columns = cur.execute("PRAGMA table_info(rocket_details_2)").fetchall()
```

```

        new_rocket_d2 = []
        for column in rocket_d2_columns:
            if column["name"] in request.form.keys():
                new_rocket_d2 += [request.form[column["name"]]]

        cur.execute(f'INSERT INTO rocket_details_2 VALUES ({",".join("?" * len(rocket_d2_columns))})', new_rocket_d2)
        con.commit()
def add_rocket_image(request):
    with sqlite3.connect(db_location) as con:
```

After these steps, the user is redirected to the rocket viewing page where they can see their addition.

Rocket Measurements Editing Modal When a user wishes to edit a rocket's measurements, they open the "Update Measurements" modal as described in the User Guide. This modal contains a HTML form based on the same WTForm as the measurement addition modal. The only difference is that this modal was generated with a pre-entered ID for the relevant rocket as follows:

rockets.html

```

{% for rocket_d2 in rocket_d2s %}
<div class="modal fade" id="update-rocket-details-2-modal-{{ rocket_d2.rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="update-rocket-details-2-modal-label-{{ rocket_d2.rocket_id }}" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="update-rocket-details-2-modal-label-{{ rocket_d2.rocket_id }}"> Update {{rocket_d2.name}} Measurements </h5>
        </div>
        <div class="modal-body">
          <form action="/update_rocket_detail_2" method="POST" style="display: grid;">
          {{ formD2.csrf_token }}
          <div class="field" style="display: none">
              <div class="control">
                  {{ formD2.rocket_id(value=rocket_d2.rocket_id, class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="height_mt" class="form-label detail-field">Height (m):</label>
              <div class="control detail-field">
                  {{ formD2.height_mt(value=rocket_d2.height_mt, class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="height_ft" class="form-label detail-field">Height (ft):</label>
              <div class="control detail-field">
                  {{ formD2.height_ft(value=rocket_d2.height_ft, class="form-control-sm")}}
              </div>
          </div>
```

```
        <div class="field">
            <label for="diameter_mt" class="form-label detail-field">Diameter (m):</label>
            <div class="control detail-field">
                {{ formD2.diameter_mt(value=rocket_d2.diameter_mt, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="diameter_ft" class="form-label detail-field">Diameter (ft):</label>
            <div class="control detail-field">
                {{ formD2.diameter_ft(value=rocket_d2.diameter_ft, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="mass_kg" class="form-label detail-field">Mass (kg):</label>
            <div class="control detail-field">
                {{ formD2.mass_kg(value=rocket_d2.mass_kg, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="mass_lb" class="form-label detail-field">Mass (lb):</label>
            <div class="control detail-field">
                {{ formD2.mass_lb(value=rocket_d2.mass_lb, class="form-control-sm")}}
            </div>
        </div>

        <button type=submit class="btn btn-warning">Update</button>
      </form>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
      </div>
    </div>
   </div>
  </div>
{% endfor %}
```

Once they submit this form, the data is validated and passed on to the database handler as follows:

views.py

```
@views.route('/update_rocket_detail_2', methods=['POST'])
def update_rocket_detail_2():
    form = forms.RocketD2Form()
    if form.validate_on_submit():
        database.update_rocket_d2(request)
    return redirect(url_for("views.rockets"))
```

The database operation is as follows:

database.py

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()

```

```
        # Update rocket_d2
        rocket_d2_columns = cur.execute("PRAGMA table_info(rocket_details_2)").fetchall()
        rocket_d2_columns_str = ",".join([column["name"] + "=?" for column in rocket_d2_columns if column["name"] != "rocket_id"])

        rocket_d2 = []
        for column in rocket_d2_columns:
            if column["name"] in request.form.keys():
                rocket_d2 += [request.form[column["name"]]]

        rocket_id = rocket_d2[0]
        rocket_d2 = rocket_d2[1:] + [rocket_id]

        cur.execute(f'UPDATE rocket_details_2 SET {rocket_d2_columns_str} WHERE rocket_id = ?', rocket_d2)
        con.commit()

def delete_rocket(rocket_id):
```

After these steps, the user is redirected to the rocket viewing page where they can see their edit.

Rocket Measurements Deletion When a user sends a request for the deletion of a rocket's measurements, this request is handled as follows:

`views.py`

```
def delete_rocket_detail_2():
    database.delete_rocket_d2(request.args.get("rocket_id"))
    return redirect(url_for("views.rockets"))
```

This deletion is then performed on the database as follows:

`database.py`

```
        cursor = con.cursor()
        query = "DELETE FROM rocket_details_2 WHERE (rocket_id = ?)"
        cursor.execute(query, (rocket_id,))
        con.commit()
def delete_rocket_image(rocket_id):
    with sqlite3.connect(db_location) as con:
```

This data is also deleted when the parent rocket is deleted.

After these steps, the user is redirected to the rocket viewing page where they can see the result of their deletion.

2.3.5 Rocket Images

Rocket Image Viewing Page Rocket images are accesses through their own modals which are generated as follows:

`rockets.html`

```

{% for rocket_image in rocket_images %}
<div class="modal fade" id="rocket-image-modal-{{ rocket_image.rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="rocket-image-modal-label-{{ rocket_image.rocket_id }}" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
```

```
          <h5 class="modal-title" id="rocket-image-modal-label-{{ rocket_image.rocket_id }}"> {{ rocket_image.name }} Photo </h5>
        </div>
        <div class="modal-body">
            <img src="/static/rocket_images/{{ rocket_image.rocket_id }}.png" alt="Photo of {{ rocket_image.name }}.">
        </div>
        <div class="modal-footer">
          <button type="button" onclick="delete_rocket_image('{{ rocket_image.rocket_id }}')"
           class="btn btn-danger" data-dismiss="modal">Delete Photo</button>
          <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
        </div>
      </div>
    </div>
  </div>
{% endfor %}
```

Images are stored in our database as BLOBs. The image file for image modals are pulled from the database and made available in the following function as static files:

database.py

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        data = cur.execute(
            """SELECT * FROM rocket_images
            JOIN (SELECT rocket_id, name FROM rockets) AS rocket_names
            ON rocket_images.rocket_id = rocket_names.rocket_id"""
            ).fetchall()
        for row in data:
            try:
                with open("/static/rocket_images/" + row["rocket_id"] + ".png", "wb+") as image_file:
                    image_file.write(row["rocket_image"])
            except FileNotFoundError:
                with open("website/static/rocket_images/" + row["rocket_id"] + ".png", "wb+") as image_file:
                    image_file.write(row["rocket_image"])
        return data

def add_rocket(request):
```

This static file is then referenced in the image modal.

Rocket Image Adding Modal Rockets which do not have images have an image upload modal generated as follows:

rockets.html

```

{% for rocket in inexistent_image_dict %}
<div class="modal fade" id="add-rocket-image-modal-{{ rocket.rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="add-rocket-image-modal-label-{{ rocket.rocket_id }}" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="rocket-image-modal-label-{{ rocket.rocket_id }}"> Add {{rocket.name}} Photo </h5>
        </div>
```

```
        <div class="modal-body">
          <form action="/add_rocket_image" method="POST" enctype="multipart
    /form-data" style="display: grid;">
          {{ formI.csrf_token }}
          <div class="field" style="display: none">
              <div class="control">
                  {{ formI.rocket_id(value=rocket.rocket_id, class="form-
    control-sm")}}
              </div>
          </div>
          <div class="field">
              <div class="control detail-field">
                  {{ formI.rocket_image(class="form-control")}}
              </div>

          <button type=submit class="btn btn-warning">Add</button>
        </form>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-dismiss="
    modal">Close</button>
        </div>
      </div>
    </div>
  </div>
</div>
{% endfor %}
```

The form for this modal is as so:

forms.py

```
class RocketImageForm(FlaskForm):
    rocket_id = StringField()
    rocket_image = FileField(validators=[validators.DataRequired()])
```

When the user submits the image upload form, it is directed by the following function to the database handler:

views.py

```
@views.route('/add_rocket_image', methods=['POST'])
def add_rocket_image():
    form = forms.RocketImageForm()
    if form.validate_on_submit():
        database.add_rocket_image(request)
    return redirect(url_for("views.rockets"))
```

The uploaded image is sent to the database by being passed as a byte string to the SQL statement as so:

database.py

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new rocket_image

        new_rocket_image = [request.form["rocket_id"]]

        data = request.files["rocket_image"].read()
        print(data)

        new_rocket_image += [data]

```

```
        cur.execute(f'INSERT INTO rocket_images VALUES (?, ?)',
    new_rocket_image)
        con.commit()

def update_rocket(request):
```

After these steps, the user is redirected to the rocket viewing page where they can see the result of their addition.

Rocket Image Deletion When a user sends a request to delete a rocket's image, this request is handled by the following function:

views.py

```
@views.route('/delete_rocket_image', methods=['GET'])
def delete_rocket_image():
    database.delete_rocket_image(request.args.get("rocket_id"))
    return redirect(url_for("views.rockets"))
```

The specified rocket is deleted as so:

database.py

```
        cursor = con.cursor()
        query = "DELETE FROM rocket_images WHERE (rocket_id = ?)"
        cursor.execute(query, (rocket_id,))
        con.commit()

# Ships
```

Deletion on a rocket image is also performed when a rocket itself is deleted.

After these steps, the user is redirected to the rocket viewing page where they can see the result of their deletion.

2.3.6 Launchpads Viewing Page

The launchpad viewing page only has viewing functionality. When a user requests the launchpads page, the request is handled as follows:

views.py

```
@views.route('/launchpads', methods=['GET'])
def launchpads():
    launchpad_data = database.get_launchpads()
    return render_template("launchpads.html", launchpads=launchpad_data)
```

Launchpad information is pulled from the database as follows:

database.py

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            "SELECT * FROM launchpads ORDER BY name"
        )

# Payloads
```

This data is then presented in a table defined as so:

launchpads.html

```
<table class = "table table-hover table-sm table-bordered">
    <thead>
        <tr class = "table-active">
            <th>Name</th>
```

```
            <th>Full Name</th>
            <th>Status</th>
            <th>Locality</th>
            <th>Region</th>
            <th>Timezone</th>
            <th>Latitude</th>
            <th>Longitude</th>
        </tr>
    </thead>
    <tbody>
        {% for launchpad in launchpads %}
        <tr class>
            <td> {{ launchpad["name"]}} </td>
            <td> {{ launchpad["full_name"]}} </td>
            <td> {{ launchpad["status"]}} </td>
            <td> {{ launchpad["locality"]}} </td>
            <td> {{ launchpad["region"]}} </td>
            <td> {{ launchpad["timezone"]}} </td>
            <td> {{ launchpad["latitude"]}} </td>
            <td> {{ launchpad["longitude"]}} </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

2.4 Parts Owned by Alp Türkbayrak

2.4.1 Base.html Page

Bootstrap 5: The styling of this app uses 'Bootstrap5'. The following are the links to include the path to Bootstrap5 in order to use the custom classes.

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"
  integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65"
  crossorigin="anonymous">
<link
  rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
  crossorigin="anonymous"
/>
<!-- JavaScript Bundle with Popper -->
<script defer src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4"
  crossorigin="anonymous"></script>
<link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
```

Navbar: The base.html also introduces a navbar at the top of each page along with a small svg rocket icon.

```
<nav class="navbar navbar-expand-lg navbar-light bg-warning" style="padding-left: 10em">
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="#navbar"
  >
    <span class="navbar-toggler-icon"></span>
  </button>
  <svg xmlns="http://www.w3.org/2000/svg" width="25" height="25" fill="currentColor" class="bi bi-rocket" viewBox="0 0 16 16">
    <path d="M8 8c.828 0 1.5-.895 1.5-2S8.828 4 8 4s-1.5.895-1.5 2S7.172 8 8 8Z"/>
    <path d="M11.953 8.81c-.195-3.388-.968-5.507-1.777-6.819C9.707 1.233 9.23.751 8.857.454a3.495 3.495 0 0 0-.463-.315A2.19 2.19 0 0 0 8.25.064.546.546 0 0 0 8 0a.549.549 0 0 0-.266.073 2.312 2.312 0 0 0-.142.08 3.67 3.67 0 0 0-.459.33c-.37.308-.844.803-1.31 1.57-.805 1.322-1.577 3.433-1.774 6.756l-1.497 1.826-.004.005A2.5 2.5 0 0 0 2 12.202V15.5a.5.5 0 0 0 .9.3l1.125-1.5c.166-.222.42-.4.752-.57.214-.108.414-.192.625-.281l.198-.084c.7.428 1.55.635 2.4.635.85 0 1.7-.207 2.4-.635.067.03.132.056.196.083.213.09.413.174.627.282.332.17.586.348.752.57l1.125 1.5a.5.5 0 0 0 .9-.3v-3.298a2.5 2.5 0 0 0-.548-1.562l-1.499-1.83ZM12 10.445v.055c0 .866-.284 1.585-.75 2.14.146.064.292.13.425.199.39.197.8.46 1.1.86L13 14v-1.798a1.5 1.5 0 0 0-.327-.935L12 10.445ZM4.75 12.64C4.284 12.085 4 11.366 4 10.5v-.054l-.673.82a1.5 1.5 0 0 0-.327.936V14l.225-.3c.3-.4.71-.664 1.1-.861.133-.068.279-.135.425-.199ZM8.009 1.073c.063.04.14.094.226.163.284.226.683.621 1.09 1.28C10.137 3.836 11 6.237
```

```
11 10.5c0 .858-.374 1.48-.943 1.893C9.517 12.786 8.781 13 8 13c-.781 0-1.517-.214-2.057-.607C5.373 11.979 5 11.358 5 10.5c0-4.182.86-6.586 1.677-7.928.409-.67.81-1.082 1.096-1.32.09-.076.17-.135.236-.18Z"/>
      <path d="M9.479 14.361c-.48.093-.98.139-1.479.139-.5 0-.999-.046-1.479-.139L7.6 15.8a.5.5 0 0 0 .8 0l1.079-1.439Z"/>
    </svg>
    <div class="collapse navbar-collapse" id="navbar">
      <div class="navbar-nav">
        <a class="nav-item nav-link" id="home" href="/home">Home</a>
        <a class="nav-item nav-link" id="launches" href="/launches">Launches</a>
        <a class="nav-item nav-link" id="launchpads" href="/launchpads">Launchpads</a>
        <a class="nav-item nav-link" id="rockets" href="/rockets">Rockets</a>
        <a class="nav-item nav-link" id="ships" href="/ships">Ships</a>
        <a class="nav-item nav-link" id="payloads" href="/payloads">Payloads</a>
        <a class="nav-item nav-link" id="cores" href="/cores">Cores</a>
        <a class="nav-item nav-link" id="capsules" href="/capsules">Capsules</a>

        {% if not current_user.is_authenticated %}
          <a class="nav-item nav-link pad-left" href="{{ url_for('views.login') }}">Login</a>
        {% else %}
          <p class="nav-item nav-link pad-left" style="margin: 0;">You are logged in as {{ current_user.username }}.</p>
          <a class="nav-item nav-link pad-left" href="{{ url_for('views.logout') }}">Logout</a>
        {% endif %}

      </div>
    </div>
  </nav>
```

2.4.2 Home.html Page

Home page is the simplest page in the app. It only includes 3 card elements to get the user started. It utilises lists and bootstrap's grid too.

```

<div class = "container">
    <div class = "row">
        <div class ="col">
            <div class="card" style="width: 20rem;">
                    <div class="card-body">
                    <h5 class="card-title">Search</h5>
                    <p class="card-text">You can make extensive searches about space missions conducted by SpaceX with our website!
                        You can get information on: </p>
                    </div>
                    <ul class="list-group list-group-flush">
                    <li class="list-group-item"><ul><li>Launches</li></ul></li>
                    <li class="list-group-item"><ul><li>Launchpads</li></ul></li>
                    <li class="list-group-item"><ul><li>Rockets</li></ul></li>
```

```
                    <li class="list-group-item"><ul><li>Ships</li></ul></li>
                    <li class="list-group-item"><ul><li>Payloads</li></ul></li>
                    <li class="list-group-item"><ul><li>Cores</li></ul></li>
                    <li class="list-group-item"><ul><li>Capsules</li></ul></li>

                    </ul>
            </div>
        </div>
        <div class = "col">
            <div class="card" style="width: 20rem;">
                <div class="card-body">
                <h5 class="card-title">Edit</h5>
                <p class="card-text">You can change our database with the simple tools we provide you!
                    Here is a list of what you can do: </p>
                </div>
                <ul class="list-group list-group-flush">
                <li class="list-group-item"><ul><li>Create entries</li></ul></li>
                <li class="list-group-item"><ul><li>Update entries</li></ul></li>
                <li class="list-group-item"><ul><li>Delete entries</li></ul></li>
                </ul>
            </div>
        </div>

        <div class = "col">
            <div class="card" style="width: 20rem;">
                <div class="card-body">
                <h5 class="card-title">Get Started</h5>
                <p class="card-text">To get started, click one of the tabs on the navigation bar above, that's all!</p>
                </div>
            </div>
            <br/>
            <img src="static/rocket_images/rocket_icon.png" style="width: 22rem;">
        </div>
    </div>
</div>
```

2.4.3 Ship Technics

The database interaction to get payload information is as follows in database.py:

```
def get_ship_d1():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            """SELECT * FROM ship_details_1
            JOIN (SELECT ship_id, name FROM ships) AS ship_names
            ON ship_details_1.ship_id = ship_names.ship_id"""
            ).fetchall()
```

Adding Ship Technics: WTForms library is used for the adding operation. The library consists of different fields for input and equivalent for <input> tag. The technics information form consists of these items:

```
class ShipD1Form(FlaskForm):
    ship_id = StringField()
    model = StringField()
    primary_role = StringField()
    secondary_role = StringField()
    imo = IntegerField()
    mmsi = IntegerField()
    abs = IntegerField()
```

The view function to add Technics information in views.py:

```
@views.route('/add_ship_detail_2', methods=['POST'])
def add_ship_detail_2():
    database.add_ship_d2(request)
    return redirect(url_for("views.ships"))
```

And the SQL statement for adding Technics information in database.py:

```
def add_ship_d1(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new ship_d1
        ship_d1_columns = cur.execute("PRAGMA table_info(ship_details_1)").
    fetchall()

        new_ship_d1 = []
        for column in ship_d1_columns:
            if column["name"] in request.form.keys():
                new_ship_d1 += [request.form[column["name"]]]

        cur.execute(f'INSERT INTO ship_details_1 VALUES ({",".join("?" *
    len(ship_d1_columns))})', new_ship_d1)
        con.commit()
```

Updating Ship Technics: The same form is used for the update function as well. The view function to update Technics information in views.py:

```
@views.route('/update_ship_detail_1', methods=['POST'])
def update_ship_detail_1():
    database.update_ship_d1(request)
```

And the SQL statement for updating Technics information in database.py:

```
def update_ship_d1(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update ship_d1
        ship_d1_columns = cur.execute("PRAGMA table_info(ship_details_1)").
    fetchall()
        ship_d1_columns_str = ",".join([column["name"] + "=?" for column in
     ship_d1_columns if column["name"] != "ship_id"])


        ship_d1 = []
        for column in ship_d1_columns:
```

```
            if column["name"] in request.form.keys():
                ship_d1 += [request.form[column["name"]]]

        print("SHIP D1 COL STR: " + ship_d1_columns_str)
        ship_id = ship_d1[0]
        ship_d1 = ship_d1[1:] + [ship_id]

        cur.execute(f'UPDATE ship_details_1 SET {ship_d1_columns_str} WHERE
    ship_id = ?', ship_d1)
        con.commit()
```

Deleting Ship Technics: The view function to delete Technics information in views.py:

```
@views.route('/delete_ship_detail_1', methods=['GET'])
def delete_ship_detail_1():
    database.delete_ship_d1(request.args.get("ship_id"))
    return redirect(url_for("views.ships"))
```

SQL statement for deleting Technics information in database.py:

```
def delete_ship_d1(ship_id):
    with sqlite3.connect(db_location) as con:
        cursor = con.cursor()
        query = "DELETE FROM ship_details_1 WHERE (ship_id = ?)"
        cursor.execute(query, (ship_id,))
        con.commit()
```

And the js function code for deleting Technics information in index.js:

```
function delete_ship_d1(ship_id) {
    document.location = '/delete_ship_detail_1?ship_id=' + ship_id
}
```

2.4.4 Payloads Page

Displaying Payloads: Payloads Page is where the users perform all actions related to payloads. This page is served via the following function in views.py:

```
@views.route('/payloads', methods=['GET'])
def payloads():
    payload_data = database.get_payloads()
    return render_template("payloads.html", payloads=payload_data, formM=
   forms.PayloadForm())
```

The mentioned form for payloads is defined as follows in forms.py:

```
class PayloadForm(FlaskForm):
    payload_id = StringField()
    name = StringField()
    type = StringField()
    reused = SelectField(choices=[("True", "Yes"), ("False", "No")])
    manufacturers = StringField()
    mass_kg = IntegerField()
    mass_lb = IntegerField()
    orbit = StringField()
    reference_system = StringField()
    regime = StringField()
```

The database interaction to get payload information is as follows in database.py:

```
def get_payloads():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
```

```
        cur = con.cursor()
        return cur.execute(
            "SELECT * FROM payloads ORDER BY name ASC"
            ).fetchall()
```

The data is then returned in a table generated using Jinja Templating Language in payloads.html:

```
<table class = "table table-hover table-sm">
    <thead>
        <tr class = "table-active">
            <th>Name</th>
            <th>Type</th>
            <th>Is Reused</th>
            <th>Manufacturers</th>
            <th>Mass (kg)</th>
            <th>Mass (lb)</th>
            <th>Orbit</th>
            <th>Reference System</th>
            <th>Regime</th>
            <th colspan="2">Editing</th>
        </tr>
    </thead>
    <tbody>
        {% for payload in payloads %}
        <tr class>
            <td>
                {{payload["name"]}}
            </td>
            <td>
                {{payload["type"]}}
            </td>
            <td>
                {{"No" if payload["reused"] == "False" else "Yes"}}
            </td>
            <td>
                {{"-" if payload["manufacturers"] == None else payload["manufacturers"]}}
            </td>
            <td>
                {{"-" if payload["mass_kg"] == None else payload["mass_kg"]}}
            </td>
            <td>
                {{"-" if payload["mass_lb"] == None else payload["mass_lb"]}}
            </td>
            <td>
                {{"-" if payload["orbit"] == None else payload["orbit"]}}
            </td>
            <td>
                {{"-" if payload["reference_system"] == None else payload["reference_system"]}}
            </td>
            <td>
                {{"-" if payload["regime"] == None else payload["regime"]}}
            </td>
            <td>
                <button type="button" class="btn btn-warning btn-sm" data-toggle="modal" data-target="#update-payload-modal-{{ payload.payload_id }}">
                  Update
                </button>
```

```
            </td>
            <td>
                {% if current_user.is_admin %}
                  <button type="button" class="btn btn-danger btn-sm" data-toggle="modal"
                  onclick="delete_payload('{{ payload.payload_id }}')">
                    Delete
                  </button>
                {% endif %}
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

Adding Payloads: When the user wishes to add a new payload to the database, they click the ”New Payload” button defined as follows in payloads.html:

```
<button type="button" class="btn btn-warning" data-toggle="modal" data-target="#add-payload-modal" style="position: fixed; bottom: 5em; right: 2.5em;">
    New Payload
</button>
```

Doing so, opens up a modal defined as follows in payloads.html:

```
<div class="modal fade" id="add-payload-modal" tabindex="-1" role="dialog" aria-labelledby="add-payload-modal-label" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="add-payload-modal-label">Add Payload</h5>
                <button type="button" class="btn-close" data-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body"><form action="/add_payload" method="POST" style="display: grid;">
                {{ formM.csrf_token }}
                <div class="field">
                    <label for="name" class="form-label detail-field">Name: </label>
                    <div class="control detail-field">
                        {{ formM.name(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
                    <label for="type" class="form-label detail-field">Type: </label>
                    <div class="control detail-field">
                        {{ formM.type(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
                    <label for="reused" class="form-label detail-field">Reused?:</label>
                    <div class="control detail-field">
                        {{ formM.reused(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
```

```
                    <label for="manufacturers" class="form-label detail-field">Manufacturers:</label>
                    <div class="control detail-field">
                        {{ formM.manufacturers(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
                    <label for="mass_kg" class="form-label detail-field">Mass (kg):</label>
                    <div class="control detail-field">
                        {{ formM.mass_kg(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
                    <label for="mass_lb" class="form-label detail-field">Mass (lb):</label>
                    <div class="control detail-field">
                        {{ formM.mass_lb(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
                    <label for="orbit" class="form-label detail-field">Orbit:</label>
                    <div class="control detail-field">
                        {{ formM.orbit(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
                    <label for="reference_system" class="form-label detail-field">Reference System:</label>
                    <div class="control detail-field">
                        {{ formM.reference_system(class="form-control-sm")}}
                    </div>
                </div>
                <div class="field">
                    <label for="regime" class="form-label detail-field">Regime:</label>
                    <div class="control detail-field">
                        {{ formM.regime(class="form-control-sm")}}
                    </div>
                </div>

                <button type="submit" class="btn btn-warning">Add</button>
            </form>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
            </div>
        </div>
    </div>
</div>
```

This modal includes a form which reads user input and feeds it to views.py function to add payloads. Add payloads function is implemented as follows in views.py:

```
@views.route('/add_payload', methods=['POST'])
@login_required
def add_payload():
    database.add_payload(request)
```

```
    return redirect(url_for("views.payloads" ))
```

After the form is validated in views.py, it gets sent to the database. The SQL statement to add payload information is defined as follows in database.py:

```
def add_payload(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new payload
        payload_columns = cur.execute("PRAGMA table_info(payloads)").fetchall()

        new_payload = [str(current_app.config["payload_id"])]  # Get next ID
        current_app.config["payload_id"] += 1
        for column in payload_columns:
            if column["name"] in request.form.keys():
                new_payload += [request.form[column["name"]]]

        cur.execute(f'INSERT INTO payloads VALUES ({",".join("?" * len(payload_columns))})', new_payload)
        con.commit()
```

Here, it is important to note that due to our data not having integer IDs for data already present in it (it has strings instead), we had to generate IDs ourselves sequentially for any new additions we were to make. After these steps, the user is redirected to the rocket viewing page where they can see their addition.

Updating Payloads: When the user wishes to edit a rocket, they open a modal created with the same WTForm as rocket addition but with the payload ID already entered in the template using Jinja.

```
{% for payload in payloads %}

<div class="modal fade" id="update-payload-modal-{{ payload.payload_id }}" tabindex="-1" role="dialog" aria-labelledby="update-payload-modal-label-{{ payload.payload_id }}" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="payload-modal-label-{{ payload.payload_id }}"> Update {{ payload.name }} </h5>
        </div>
        <div class="modal-body"><form action="/update_payload" method="POST" style="display: grid;">
          {{ formM.csrf_token }}
          <div class="field" style="display: none">
              <div class="control">
                  {{ formM.payload_id(value=payload.payload_id, class="form-control-sm")}}
              </div>
          </div>
          <div class="field">
              <label for="name" class="form-label detail-field">Name:</label>
              <div class="control detail-field">
                  {{ formM.name(value=payload.name,class="form-control-sm")}}
              </div>
```

```
21          </div>
22          <div class="field">
23              <label for="type" class="form-label detail-field">Type:</
    label>
24              <div class="control detail-field">
25                  {{ formM.type(value=payload.type,class="form-control-sm")
    }}
26              </div>
27          </div>
28          <div class="field">
29              <label for="reused" class="form-label detail-field">Reused?:<
    /label>
30              <div class="control detail-field">
31                  <!-- {{ formM.reused(value=payload.reused,class="form-
    control-sm")}} -->
32                  <select class="form-control-sm" id="reused" name="reused"
     value="True">
33                    {% if payload.reused == "True" %}
34                      <option value="True" selected>Yes</option>
35                      <option value="False">No</option>
36                    {% else %}
37                    <option value="True">Yes</option>
38                    <option value="False" selected>No</option>
39                    {% endif %}
40                  </select>
41              </div>
42          </div>
43          <div class="field">
44              <label for="manufacturers" class="form-label detail-field">
    Manufacturers:</label>
45              <div class="control detail-field">
46                  {{ formM.manufacturers(value=payload.manufacturers,class=
    "form-control-sm")}}
47              </div>
48          </div>
49          <div class="field">
50              <label for="mass_kg" class="form-label detail-field">Mass (kg
    ):</label>
51              <div class="control detail-field">
52                  {{ formM.mass_kg(value=payload.mass_kg,class="form-
    control-sm")}}
53              </div>
54          </div>
55          <div class="field">
56            <label for="mass_lb" class="form-label detail-field">Mass (lb):
    </label>
57            <div class="control detail-field">
58                {{ formM.mass_lb(value=payload.mass_lb,class="form-control-
    sm")}}
59            </div>
60        </div>
61        <div class="field">
62            <label for="orbit" class="form-label detail-field">Orbit:</
    label>
63            <div class="control detail-field">
64                {{ formM.orbit(value=payload.orbit,class="form-control-sm")
    }}
65            </div>
66        </div>
67        <div class="field">
```

```
            <label for="reference_system" class="form-label detail-field">Reference System:</label>
            <div class="control detail-field">
                {{ formM.reference_system(value=payload.reference_system, class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="regime" class="form-label detail-field">Regime:</label>
            <div class="control detail-field">
                {{ formM.regime(value=payload.regime,class="form-control-sm")}}
            </div>
        </div>

          <button type=submit class="btn btn-warning">Update</button>
        </form>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
        </div>
      </div>
    </div>
</div>
{% endfor %}
```

When this form is submitted, views.py handles it as follows:

```
@views.route("/update_payload", methods=["POST"])
def update_payload():
    database.update_payload(request)
    return redirect(url_for("views.payloads"))
```

The database operations are as follows in database.py:

```
def update_payload(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update payload
        payload_columns = cur.execute("PRAGMA table_info(payloads)").fetchall()
        payload_columns_str = ",".join([column["name"] + "=?" for column in payload_columns if column["name"] != "payload_id"])

        payload = []
        for column in payload_columns:
            if column["name"] in request.form.keys():
                payload += [request.form[column["name"]]]

        payload_id = payload[0]
        payload = payload[1:] + [payload_id]

        cur.execute(f'UPDATE payloads SET {payload_columns_str} WHERE payload_id = ?', payload)
        con.commit()
```

After these steps, the user is redirected to the payloads page where they can see their edit.

Deleting Payloads: When an admin user clicks the delete button of a payload, they send a request with the payload's ID. This is then processed by the following function in index.js:

This then makes a call for the views.py. Users can only delete payloads if they are logged in with an admin account. The delete button is only shown in this case while the backend checks this condition in views.py:

```
@views.route('/delete_payload', methods=['GET'])
@login_required
def delete_payload():
    if current_user.is_admin:
        database.delete_payload(request.args.get("payload_id"))
    else:
        flash("Please do not poke around the exhibit.")
    return redirect(url_for("views.payloads"))
```

Then, the SQL statement in database.py is executed:

```
def delete_payload(payload_id):
    with sqlite3.connect(db_location) as con:
        cursor = con.cursor()
        print("USER IS ADMIN, DELETE PAYLOAD")
        #cursor.execute("PRAGMA foreign_keys=ON")  # This allows for cascading to details
        query = "DELETE FROM payloads WHERE (payload_id = ?)"
        cursor.execute(query, (payload_id,))
        con.commit()
```

After these steps, the user is redirected to the payloads page where the deleted row is no longer displayed.

Searching Payloads: When the user wants to make a search through the payloads in our database, they can click the "Search in Payloads" button on lower right corner of the screen defined as follows in payloads.html:

```
<button type="button" class="btn btn-warning" data-toggle="modal" data-target="#filter-payload-modal" style="position: fixed; bottom: 2.5em; right: 2.5em;">
    Search in Payloads
</button>
```

Doing so opens up a modal with a form to take input from the user to filter. The modal and the form are defined as follows in payloads.html:

```
<div class="modal fade" id="filter-payload-modal" tabindex="-1" aria-labelledby="filter-payload-modal-label" aria-hidden="true">
    <div class="modal-dialog modal-sm">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="filter-payload-modal-label">Search</h5>
                <button type="button" class="btn-close" data-dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body"><form action="/payloads_filtered" method="POST">
                    <label for="fname" class="form-label">Name:</label>
                    <input type="text" class="form-control" name="fname">

                    <label for="ftype" class="form-label">Type:</label>
                    <input type="text" class="form-control" name="ftype">
```

```

                    <label for="freused" class="form-label">Reused:</label>
                    <select class="form-select" name="freused" aria-label="Default Select">
                        <option value=""> Any </option>
                        <option value="True"> Yes </option>
                        <option value="False"> No </option>
                    </select>

                    <label for="fmanufacturers" class="form-label">Manufacturers:</label>
                    <input type="text" class="form-control" name="fmanufacturers">

                    <label for="forbit" class="form-label">Orbit:</label>
                    <input type="text" class="form-control" name="forbit">

                    <label for="freference_system" class="form-label">Reference System:</label>
                    <input type="text" class="form-control" name="freference_system">

                    <label for="fregime" class="form-label">Regime:</label>
                    <input type="text" class="form-control" name="fregime">

                    <div class="modal-footer">
                        <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
                        <button type="submit" class = "btn btn-warning">Search</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
```

Filling in this form sends the information to views.py where update_payloads function handles it as follows in views.py:

```
@views.route('/payloads_filtered', methods=['GET', 'POST'])
def payloads_filtered():
    filter_data = database.filter_payloads(request)
    return render_template("payloads.html", payloads=filter_data, formM=forms.PayloadForm())
```

This then executes the function with the SQL statements defined as follows in database.py:

```
def filter_payloads(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        print("filter_payloads executed")
        query = "SELECT * FROM payloads"
        params = []
        print(request.form.get('fname'))
        if request.form.get('fname'):
            if not params:
                query += " WHERE"
            user_name = request.form['fname']
            query += " name LIKE ?"
            param_name = "%" + user_name + "%"
```

```
15            params.append(param_name)
16
17        if request.form.get('ftype'):
18            if not params:
19                query += " WHERE"
20            else:
21                query += " AND"
22            user_type = request.form['ftype']
23            query += " type LIKE ?"
24            param_type = "%" + user_type + "%"
25            params.append(param_type)
26
27        if request.form.get('freused') != "":
28            if not params:
29                query += " WHERE"
30            else:
31                query += " AND"
32            user_reused = request.form['freused']
33            query += " reused LIKE ?"
34            param_reused = "%" + user_reused + "%"
35            params.append(param_reused)
36
37        if request.form.get('fmanufacturers'):
38            if not params:
39                query += " WHERE"
40            else:
41                query += " AND"
42            user_manufacturers = request.form['fmanufacturers']
43            query += " manufacturers LIKE ?"
44            param_manufacturers = "%" + user_manufacturers + "%"
45            params.append(param_manufacturers)
46
47        if request.form.get('forbit'):
48            if not params:
49                query += " WHERE"
50            else:
51                query += " AND"
52            user_orbit = request.form['forbit']
53            query += " orbit LIKE ?"
54            param_orbit = "%" + user_orbit + "%"
55            params.append(param_orbit)
56
57        if request.form.get('freference_system'):
58            if not params:
59                query += " WHERE"
60            else:
61                query += " AND"
62            user_reference_system = request.form['freference_system']
63            query += " reference_system LIKE ?"
64            param_reference_system = "%" + user_reference_system + "%"
65            params.append(param_reference_system)
66
67        if request.form.get('freference_system'):
68            if not params:
69                query += " WHERE"
70            else:
71                query += " AND"
72            user_regime = request.form['fregime']
73            query += " regime LIKE ?"
74            param_regime = "%" + user_regime + "%"
```

```
            params.append(param_regime)

        print(query)
        print(tuple(params))
        filter = cur.execute(query, tuple(params)).fetchall()
        return filter
```

After all the functions are executed, the user is redirected to the payloads page that only displays the filtered rows.

2.5 Parts Owned by Ahmet Metehan Yaman

2.5.1 Ships Page Displaying

The view function parts for ship in views.py:

```
@views.route('/ships', methods=['GET', 'POST'])
def ships():
    ship_data = database.get_ships()
    ship_d1_data = database.get_ship_d1()
    ship_d2_data = database.get_ship_d2()

    inexistent_ship_d1 = []
    inexistent_ship_d2 = []
    inexistent_ship_d1_dict = []
    inexistent_ship_d2_dict = []

    present = False
    for ship in ship_data:
        for ship_d1 in ship_d1_data:
            if ship["ship_id"] == ship_d1["ship_id"]:
                present = True
                break
        if not present:
            inexistent_ship_d1 += [ship["ship_id"]]
            inexistent_ship_d1_dict += [{"ship_id": ship["ship_id"], "name"
    : ship["name"]}]
        present = False

        for ship_d2 in ship_d2_data:
            if ship["ship_id"] == ship_d2["ship_id"]:
                present = True
                break
        if not present:
            inexistent_ship_d2 += [ship["ship_id"]]
            inexistent_ship_d2_dict += [{"ship_id": ship["ship_id"], "name"
    : ship["name"]}]
        present = False

    return render_template("ships.html", ships=ship_data,
        ship_d1s=ship_d1_data, ship_d2s=ship_d2_data, inexistent_d1=
    inexistent_ship_d1, inexistent_d2=inexistent_ship_d2, inexistent_d1_dict
    =inexistent_ship_d1_dict, inexistent_d2_dict=inexistent_ship_d2_dict,
        formM=forms.ShipForm(), formD1=forms.ShipD1Form(), formD2=forms.
    ShipD2Form())
```

SQL statement in database.py:

```
def get_ships_only():
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        return cur.execute(
            "SELECT * FROM ships"
            ).fetchall()
```

Jinja template for ships.html:

```
<table class = "table table-hover table-sm">
    <thead>
        <tr class = "table-active">
            <th>Name</th>
            <th>Type</th>
```

```
6            <th>Status</th>
7            <th colspan="2">Details</th>
8            <th colspan="2">Editing</th>
9        </tr>
10    </thead>
11    <tbody>
12        {% for ship in ships %}
13        <tr class>
14            <td> {{ship["name"]}} </td>
15            <td> {{ship["type"]}} </td>
16            <td> {{ "Active" if ship["active"] == "True" else "Deactive" }}
    </td>
17            <td>
18              {% if ship["ship_id"] in inexistent_d1 %}
19                <button type="button" class="btn btn-warning btn-sm" data-
    toggle="modal" data-target="#add-ship-details-1-modal-{{ ship.ship_id }}
    ">
20                Add Technics
21              </button>
22              {% else %}
23                <button type="button" class="btn btn-warning btn-sm" data-
    toggle="modal" data-target="#ship-details-1-modal-{{ ship.ship_id }}">
24                Technics
25              </button>
26              {% endif %}
27            </td>
28            <td>
29              {% if ship["ship_id"] in inexistent_d2 %}
30                <button type="button" class="btn btn-warning btn-sm" data-
    toggle="modal" data-target="#add-ship-details-2-modal-{{ ship.ship_id }}
    ">
31                Add Measurements
32              </button>
33              {% else %}
34                <button type="button" class="btn btn-warning btn-sm" data-
    toggle="modal" data-target="#ship-details-2-modal-{{ ship.ship_id }}">
35                Measurements
36              </button>
37              {% endif %}
38            </td>
39            <td>
40                <button type="button" class="btn btn-warning btn-sm" data-
    toggle="modal" data-target="#update-ship-modal-{{ ship.ship_id }}">
41                Update
42              </button>
43            </td>
44            <td>
45                {% if current_user.is_admin %}
46                  <button type="button" class="btn btn-danger btn-sm" data-
    toggle="modal"
47                  onclick="delete_ship('{{ ship.ship_id }}')">
48                    Delete
49                  </button>
50                {% endif %}
51            </td>
52
53
54        </tr>
55        {% endfor %}
56    </tbody>
```

```
</table>
```

2.5.2 Ship Adding

The view function to add ship in views.py:

```
@views.route("/add_ship", methods=["POST"])
def add_ship():
    database.add_ship(request)
    return redirect(url_for("views.ships"))
```

SQL statement for add ship in database.py:

```
def add_ship(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new ship
        ship_columns = cur.execute("PRAGMA table_info(ships)").fetchall()

        new_ship = [str(current_app.config["ship_id"])]  # Get next ID
        current_app.config["ship_id"] += 1
        for column in ship_columns:
            if column["name"] in request.form.keys():
                new_ship += [request.form[column["name"]]]

        cur.execute(f'INSERT INTO ships VALUES ({",".join("?" * len(
    ship_columns))})', new_ship)
        con.commit()
```

Button for adding new ship in ships.html:

```
<button type="button" class="btn btn-warning" data-toggle="modal" data-
    target="#add-ship-modal" style="position: fixed; bottom: 5em; right: 2.5
    em;">
  New Ship
</button>
```

Jinja template for adding ships.html:

```
<div class="modal fade" id="add-ship-modal" tabindex="-1" role="dialog"
    aria-labelledby="add-ship-modal-label" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="ship-modal-label"> Add Ship </h5>
      </div>
      <div class="modal-body"><form action="/add_ship" method="POST" style=
    "display: grid;">
        {{ formM.csrf_token }}
        <div class="field">
            <label for="name" class="form-label detail-field">Name:</label>
            <div class="control detail-field">
                {{ formM.name(class="form-control-sm")}}
            </div>
        </div>
        <div class="field">
            <label for="type" class="form-label detail-field">Type:</label>
            <div class="control detail-field">
                {{ formM.type(class="form-control-sm")}}
            </div>
        </div>
```

```
        <div class="field">
            <label for="active" class="form-label detail-field">Active:</
    label>
            <div class="control detail-field">
                {{ formM.active(class="form-control-sm")}}
            </div>
        </div>
        <button type=submit class="btn btn-warning">Add</button>
      </form>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal
    ">Close</button>
      </div>
    </div>
  </div>
</div>
```

2.5.3 Ship Deleting

The view function to delete ship in views.py:

```
@views.route('/delete_ship', methods=['GET'])
@login_required
def delete_ship():
    if current_user.is_admin:
        database.delete_ship(request.args.get("ship_id"))
    else:
        flash("Please do not poke around the exhibit.")
    return redirect(url_for("views.ships"))
```

SQL statement for deleting ship in database.py:

```
        cursor = con.cursor()
        cursor.execute("PRAGMA foreign_keys=ON")  # This allows for cascading to details
        query = "DELETE FROM ships WHERE (ship_id = ?)"
        cursor.execute(query, (ship_id,))
        con.commit()

def delete_ship_d1(ship_id):
```

And the js function code for deleting ship in index.js:

```
function delete_ship(ship_id) {
    document.location = '/delete_ship?ship_id=' + ship_id
}
```

2.5.4 Ships Updating

Wtforms library is used for updating operation. The library consists of different fields for input and equivalent for <input> tag.

```
class ShipForm(FlaskForm):
    ship_id = StringField()
    name = StringField()
    type = StringField()
    active = SelectField(choices=[("True", "Active"),("False", "Deactive")])
```

The view function to update ship in views.py:

```
@views.route("/update_ship", methods=["POST"])
def update_ship():
    database.update_ship(request)
    return redirect(url_for("views.ships"))
```

And the SQL statement for updating ship in database.py:

```
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update ship
        ship_columns = cur.execute("PRAGMA table_info(ships)").fetchall()
        ship_columns_str = ",".join([column["name"] + "=?" for column in ship_columns if column["name"] != "ship_id"])

        ship = []
        for column in ship_columns:
            if column["name"] in request.form.keys():
                ship += [request.form[column["name"]]]

        ship_id = ship[0]
        ship = ship[1:] + [ship_id]

        cur.execute(f'UPDATE ships SET {ship_columns_str} WHERE ship_id = ?', ship)
        con.commit()

def update_ship_d1(request):
```

2.5.5 Ships Filtering

Wtforms library is used for f operation. The library consists of different fields for input and equivalent for <input> tag.

```
class ShipForm(FlaskForm):
    ship_id = StringField()
    name = StringField()
    type = StringField()
    active = SelectField(choices=[("True", "Active"),("False", "Deactive")
   ])
```

The view function to filter ship in views.py:

```
@views.route('/ships_filtered', methods=['GET', 'POST'])
def ships_filtered():
    filter_data = database.filter_ships(request)
    return render_template("ships.html", ships=filter_data, formM=forms.
   ShipForm())
```

And the SQL statement for filtering ship in database.py:

```
def filter_ships(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        print("filter_ships executed")
        query = "SELECT * FROM ships"
        params = []
        print(request.form.get('fname'))
        if request.form.get('fname'):
            if not params:
                query += " WHERE"
            user_name = request.form['fname']
            query += " name LIKE ?"
            param_name = "%" + user_name + "%"
            params.append(param_name)

        if request.form.get('ftype'):
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_type = request.form['ftype']
            query += " type LIKE ?"
            param_type = "%" + user_type + "%"
            params.append(param_type)

        if request.form.get('factive') != "":
            if not params:
                query += " WHERE"
            else:
                query += " AND"
            user_active = request.form['factive']
            query += " active LIKE ?"
            param_active = "%" + user_active + "%"
            params.append(param_active)

        print(query)
        print(tuple(params))
        filter = cur.execute(query, tuple(params)).fetchall()
        return filter
```

Button for filter ships in ships.html:

```
<button type="button" class="btn btn-warning" data-toggle="modal" data-target="#filter-ship-modal" style="position: fixed; bottom: 2.5em; right: 2.5em;">
  Search in Ships
</button>
```

Jinja template for filter ships.html:

```
<div class="modal fade" id="filter-ship-modal" tabindex="-1" aria-labelledby="filter-ship-modal-label" aria-hidden="true">
  <div class="modal-dialog modal-sm">
      <div class="modal-content">
          <div class="modal-header">
              <h5 class="modal-title" id="filter-ship-modal-label">Search</h5>
              <button type="button" class="btn-close" data-dismiss="modal" aria-label="Close"></button>
          </div>
          <div class="modal-body"><form action="/ships_filtered" method="POST">
                  <label for="fname" class="form-label">Name:</label>
                  <input type="text" class="form-control" name="fname">

                  <label for="ftype" class="form-label">Type:</label>
                  <input type="text" class="form-control" name="ftype">

                  <label for="factive" class="form-label">Active:</label>
                  <select class="form-select" name="factive" aria-label="Default Select">
                      <option value=""> Any </option>
                      <option value="True"> Active </option>
                      <option value="False"> Deactive </option>
                  </select>

                  <div class="modal-footer">
                      <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
                      <button type="submit" class = "btn btn-warning">Search</button>
                  </div>
              </form>
          </div>
      </div>
  </div>
</div>
```

2.5.6 Ship Measurement Adding

Wtforms library is used for adding operation. The library consists of different fields for input and equivalent for <input> tag. The measurement information form consisted of these items:

```
class ShipD2Form(FlaskForm):
    ship_id = StringField()
    class_ = DecimalField()
    mass_kg = IntegerField()
    mass_lb = IntegerField()
    year_built = IntegerField()
    home_port = StringField()
```

The view function to add measurement information in views.py:

```
@views.route('/add_ship_detail_2', methods=['POST'])
def add_ship_detail_2():
    database.add_ship_d2(request)
    return redirect(url_for("views.ships"))
```

And the SQL statement for adding measurement information in database.py:

```
def add_ship_d2(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Add new ship_d2
        ship_d2_columns = cur.execute("PRAGMA table_info(ship_details_2)").
    fetchall()

        new_ship_d2 = []
        for column in ship_d2_columns:
            if column["name"] in request.form.keys():
                new_ship_d2 += [request.form[column["name"]]]

        cur.execute(f'INSERT INTO ship_details_2 VALUES ({",".join("?" *
    len(ship_d2_columns))})', new_ship_d2)
        con.commit()
```

2.5.7 Ship Measurement Updating

Wtforms library is used for updating operation. The library consists of different fields for input and equivalent for <input> tag.

```
class ShipD2Form(FlaskForm):
    ship_id = StringField()
    class_ = DecimalField()
    mass_kg = IntegerField()
    mass_lb = IntegerField()
    year_built = IntegerField()
    home_port = StringField()
```

The view function to update ship measurement in views.py:

```
@views.route('/update_ship_detail_2', methods=['POST'])
def update_ship_detail_2():
    database.update_ship_d2(request)
    return redirect(url_for("views.ships"))
```

And the SQL statement for updating ship measurement in database.py:

```
def update_ship_d2(request):
    with sqlite3.connect(db_location) as con:
        con.row_factory = sqlite3.Row
        cur = con.cursor()

        # Update ship_d2
        ship_d2_columns = cur.execute("PRAGMA table_info(ship_details_2)").
    fetchall()
        ship_d2_columns_str = ",".join([column["name"] + "=?" for column in
     ship_d2_columns if column["name"] != "ship_id"])

        ship_d2 = []
        for column in ship_d2_columns:
            if column["name"] in request.form.keys():
                ship_d2 += [request.form[column["name"]]]
```

```

        ship_id = ship_d2[0]
        ship_d2 = ship_d2[1:] + [ship_id]
        print(ship_d2)

        print("SHIP D2 COL STR: " + ship_d2_columns_str)
        cur.execute(f'UPDATE ship_details_2 SET {ship_d2_columns_str} WHERE
    ship_id = ?', ship_d2)
        con.commit()
```

2.5.8 Ship Measurement Deleting

The view function to delete ship measurement in views.py:

```
@views.route('/delete_ship_detail_2', methods=['GET'])
def delete_ship_detail_2():
    database.delete_ship_d2(request.args.get("ship_id"))
    return redirect(url_for("views.ships"))
```

SQL statement for deleting ship measurement in database.py:

```
def delete_ship_d2(ship_id):
    with sqlite3.connect(db_location) as con:
        cursor = con.cursor()
        query = "DELETE FROM ship_details_2 WHERE (ship_id = ?)"
        cursor.execute(query, (ship_id,))
        con.commit()
```

And the js function code for deleting ship measurement in index.js:

```
function delete_ship_d2(ship_id) {
    document.location = '/delete_ship_detail_2?ship_id=' + ship_id}
```

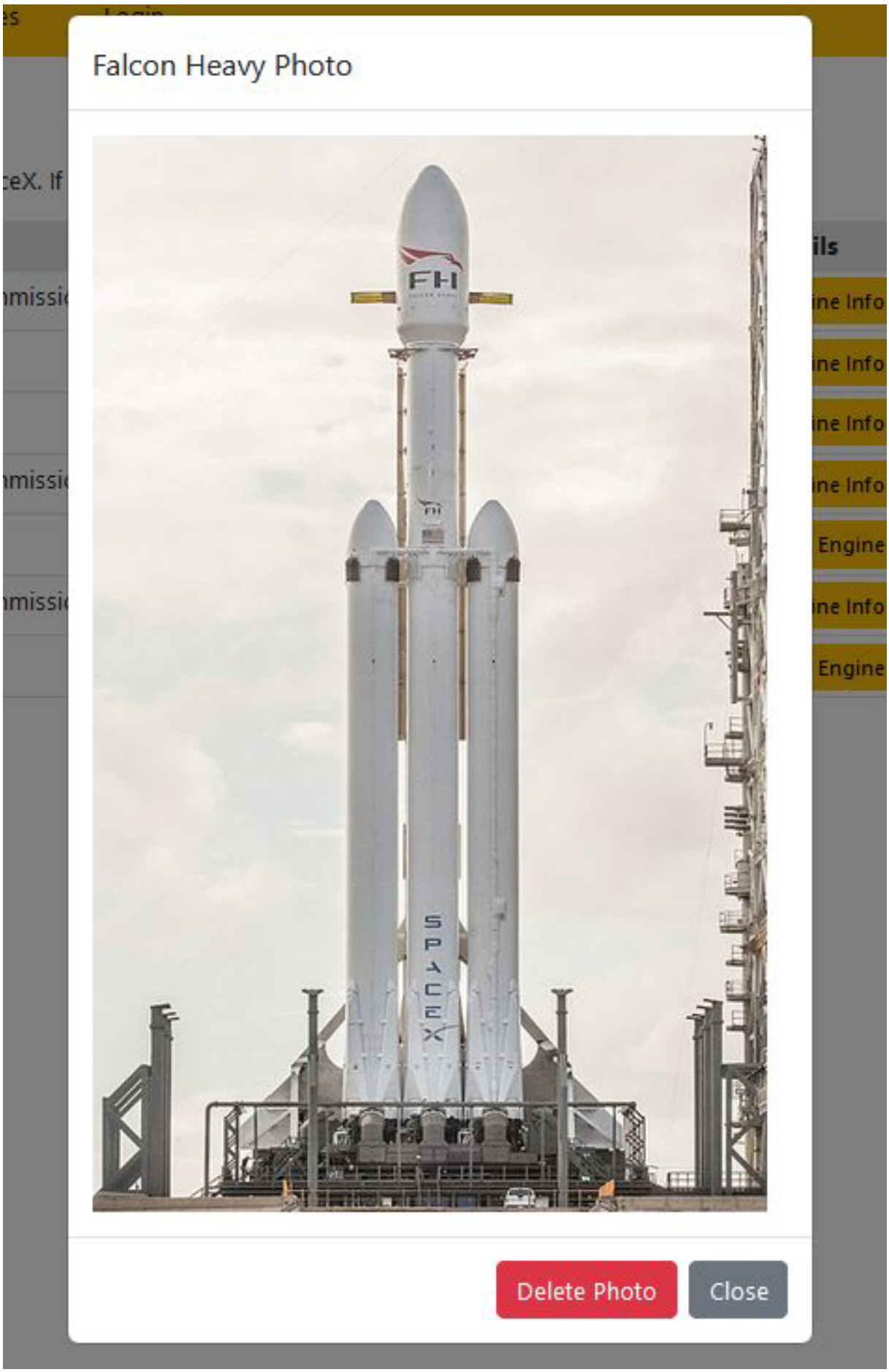


Figure 23: Rocket image view

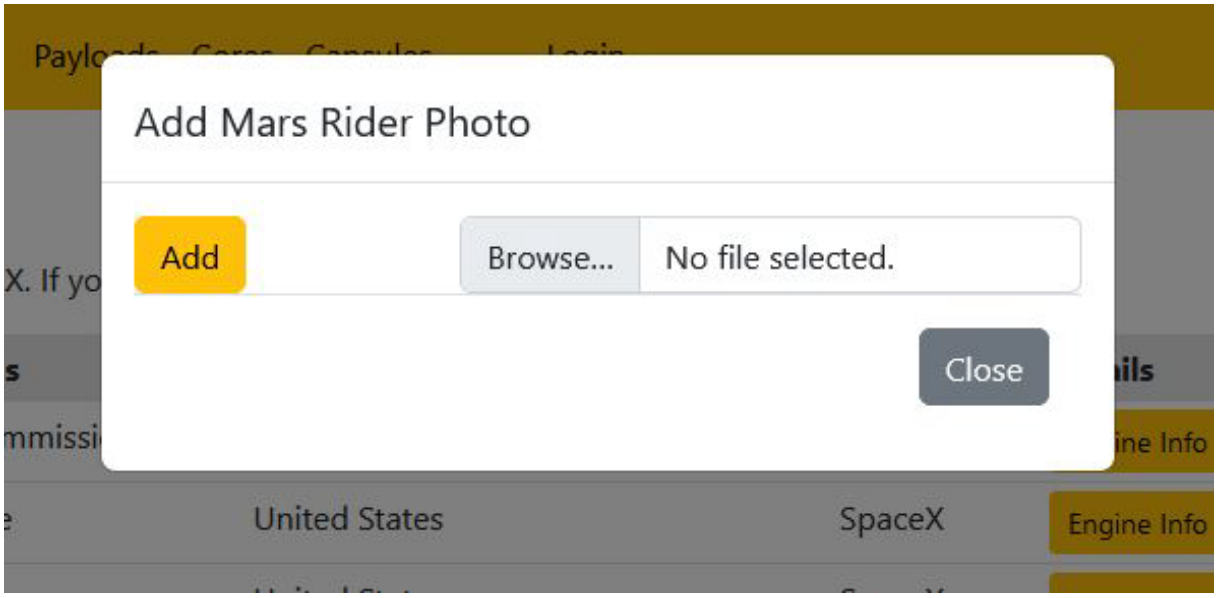
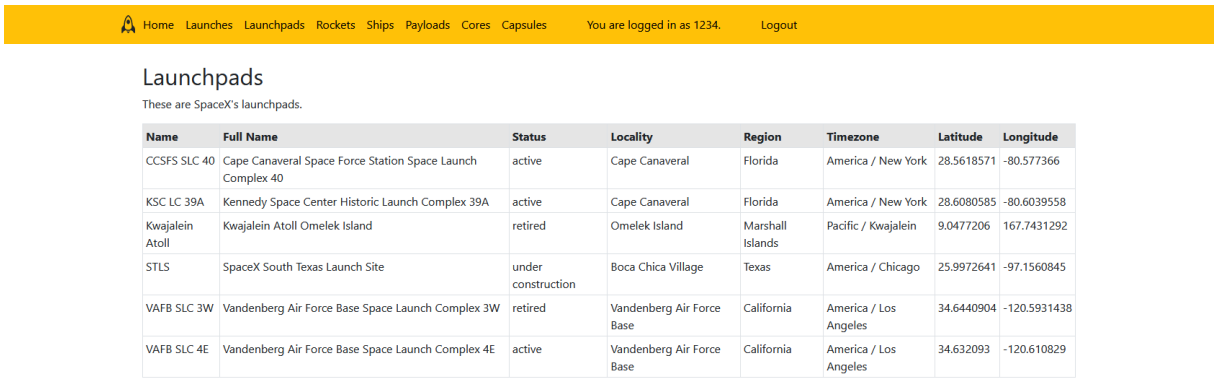


Figure 24: Rockets images adding view



Name	Full Name	Status	Locality	Region	Timezone	Latitude	Longitude
CCSFS SLC 40	Cape Canaveral Space Force Station Space Launch Complex 40	active	Cape Canaveral	Florida	America / New York	28.5618571	-80.577366
KSC LC 39A	Kennedy Space Center Historic Launch Complex 39A	active	Cape Canaveral	Florida	America / New York	28.6080585	-80.6039558
Kwajalein Atoll	Kwajalein Atoll Omelek Island	retired	Omelek Island	Marshall Islands	Pacific / Kwajalein	9.0477206	167.7431292
STLS	SpaceX South Texas Launch Site	under construction	Boca Chica Village	Texas	America / Chicago	25.9972641	-97.1560845
VAFB SLC 3W	Vandenberg Air Force Base Space Launch Complex 3W	retired	Vandenberg Air Force Base	California	America / Los Angeles	34.6440904	-120.5931438
VAFB SLC 4E	Vandenberg Air Force Base Space Launch Complex 4E	active	Vandenberg Air Force Base	California	America / Los Angeles	34.632093	-120.610829

Figure 25: Launchpads view

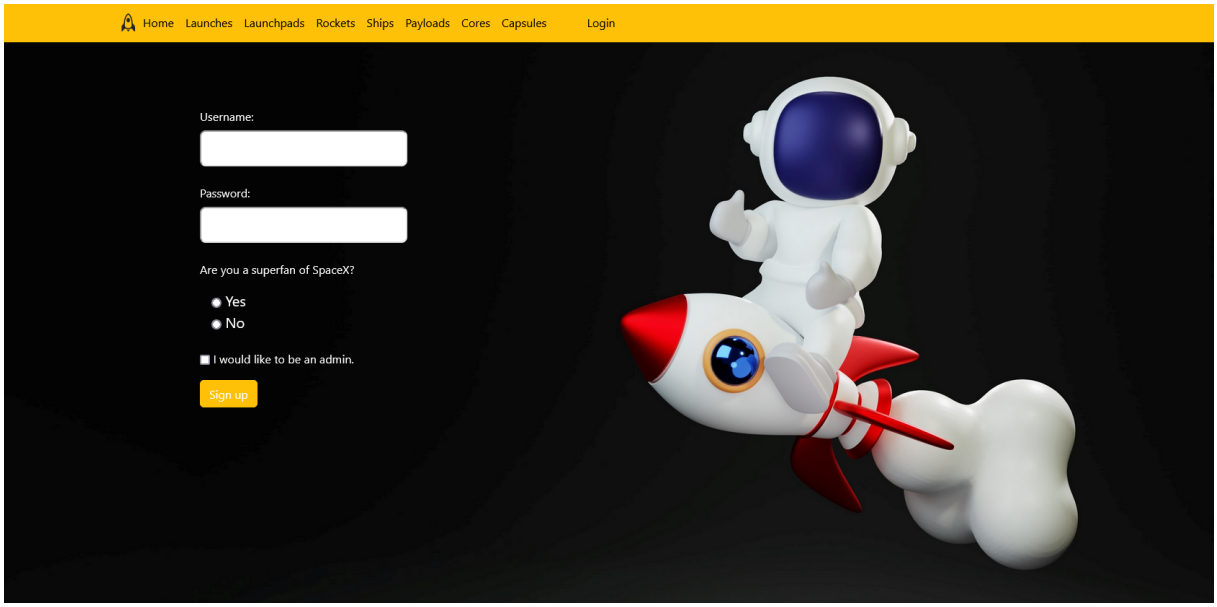


Figure 26: Signup page

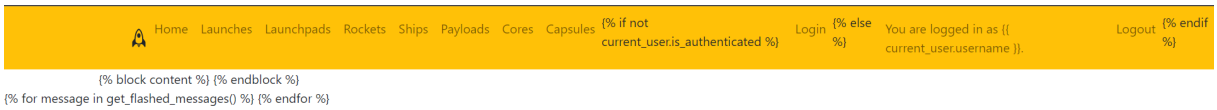


Figure 27: Base Page Design

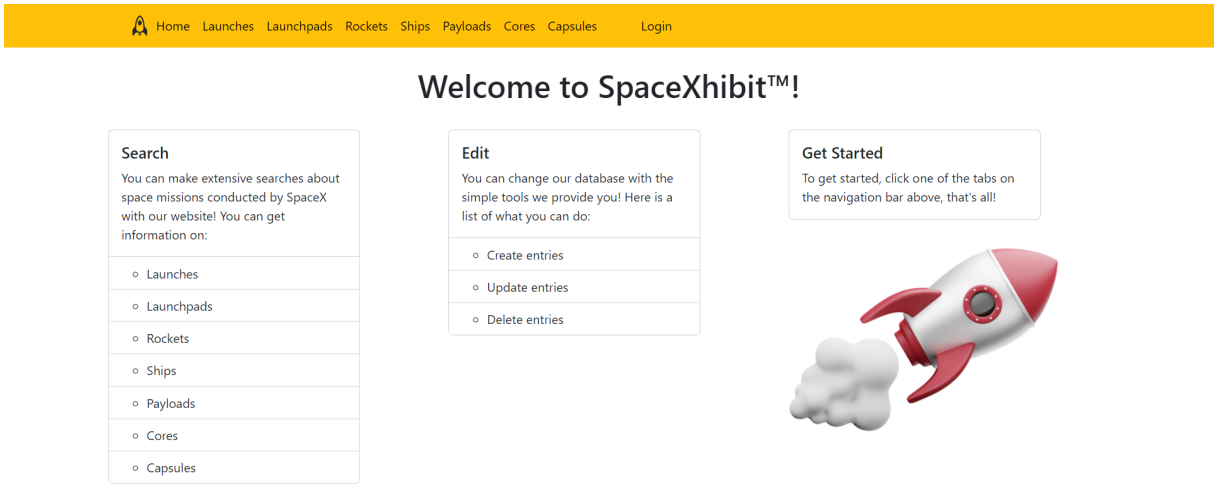


Figure 28: Home Page Design

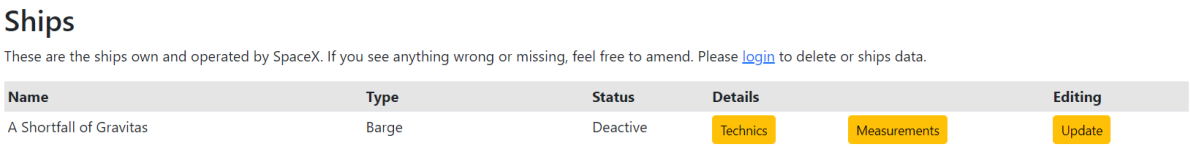


Figure 29: Technics Button on Ships Page

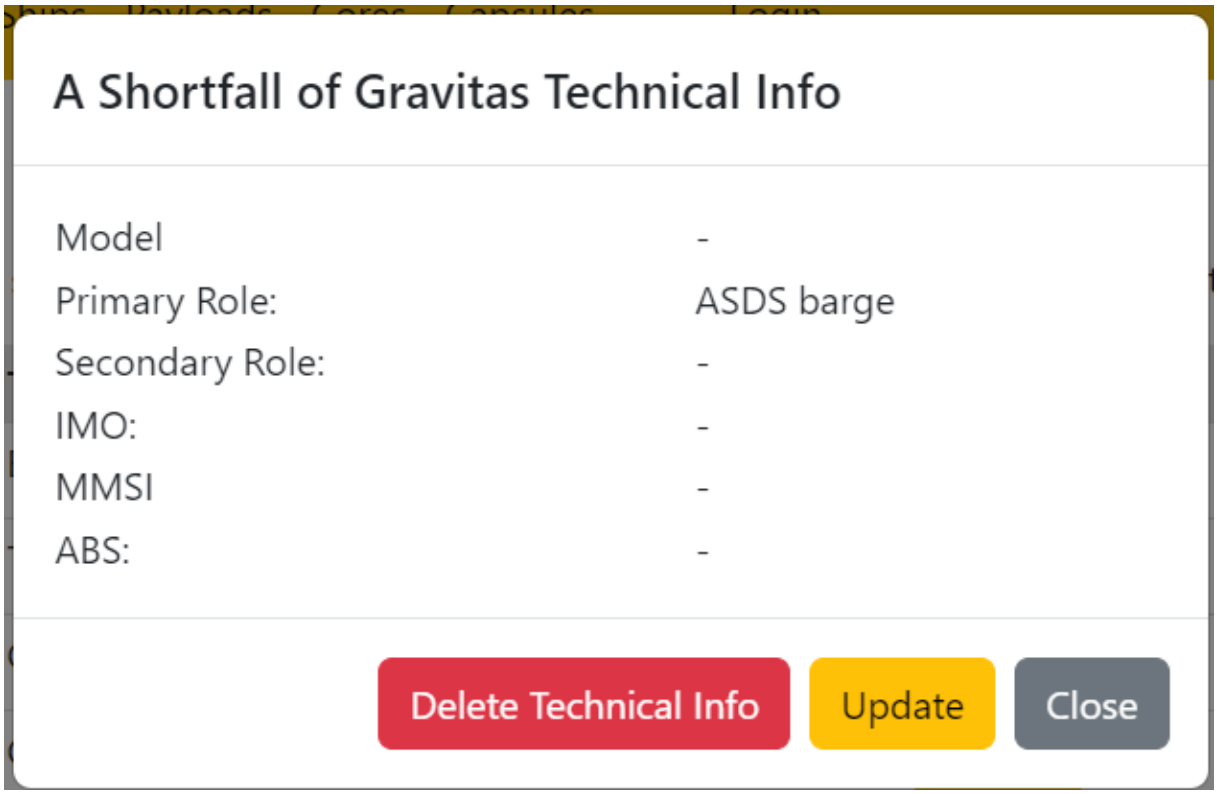


Figure 30: Technics Modal

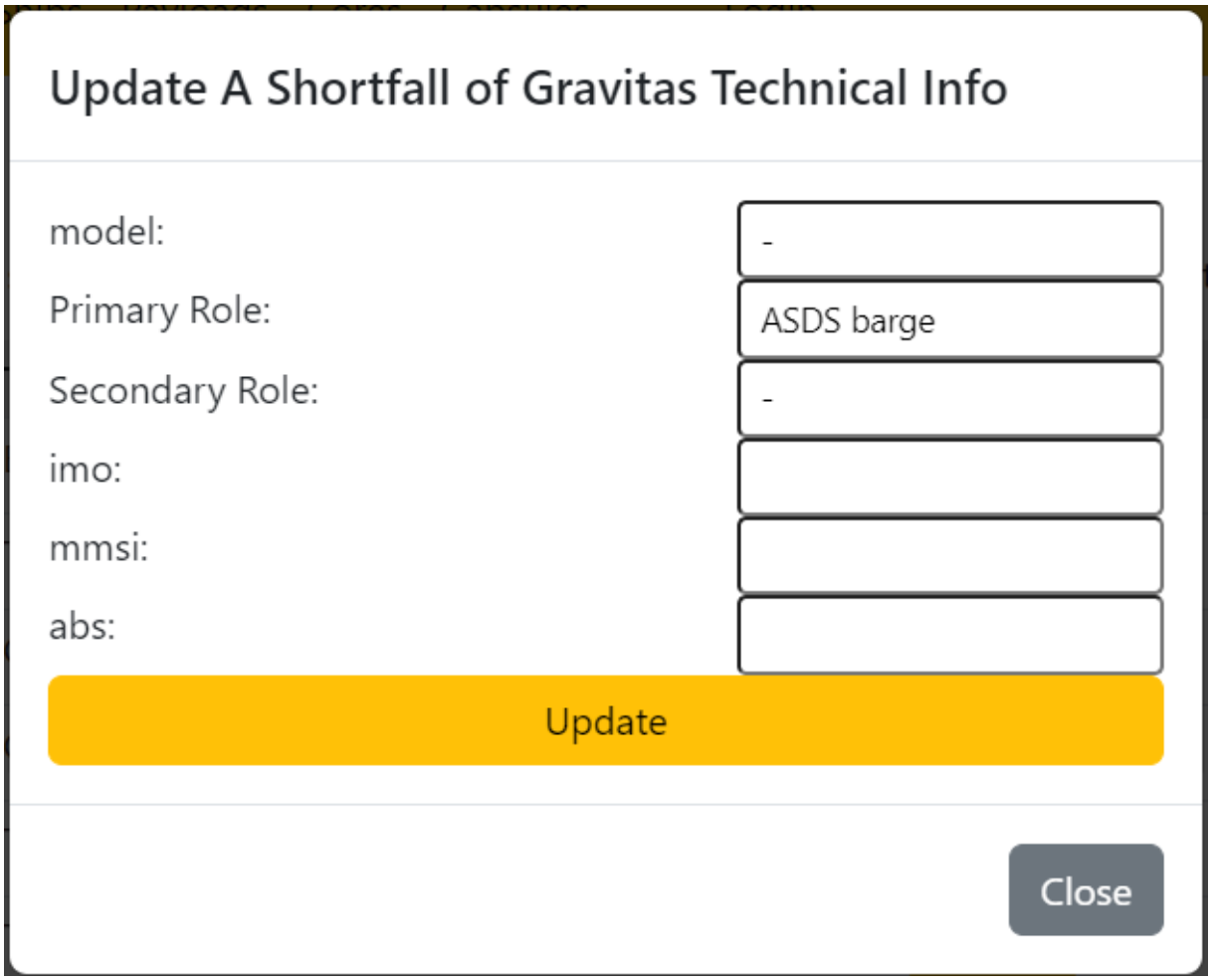
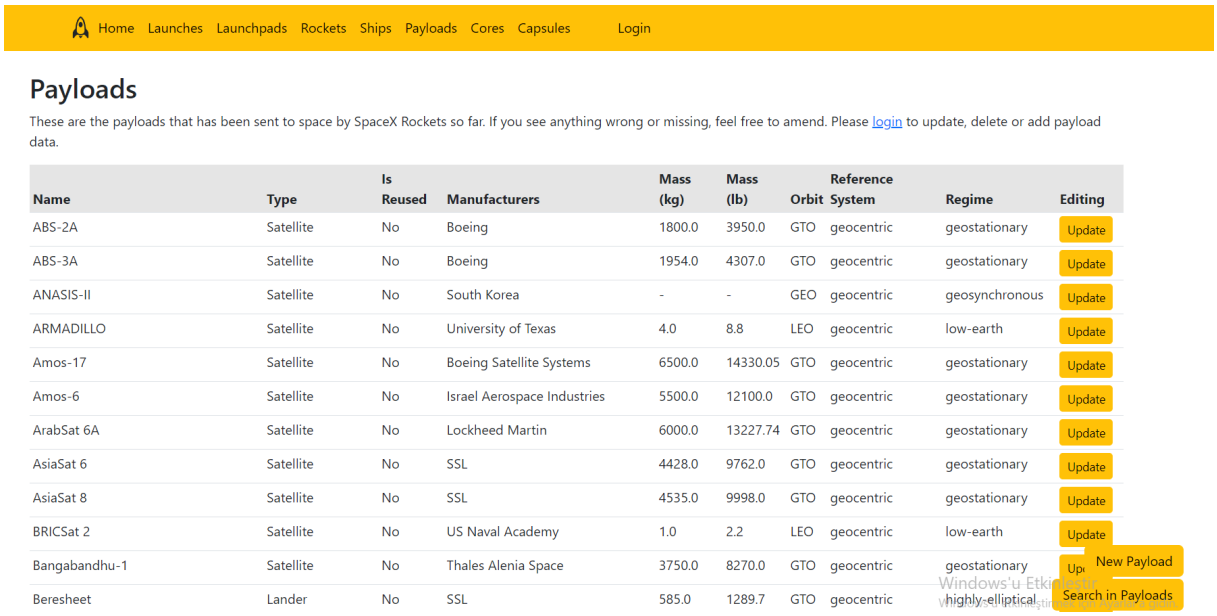


Figure 31: Technics Update



Name	Type	Is Reused	Manufacturers	Mass (kg)	Mass (lb)	Orbit	Reference System	Regime	Editing
ABS-2A	Satellite	No	Boeing	1800.0	3950.0	GTO	geocentric	geostationary	Update
ABS-3A	Satellite	No	Boeing	1954.0	4307.0	GTO	geocentric	geostationary	Update
ANASIS-II	Satellite	No	South Korea	-	-	GEO	geocentric	geosynchronous	Update
ARMADILLO	Satellite	No	University of Texas	4.0	8.8	LEO	geocentric	low-earth	Update
Amos-17	Satellite	No	Boeing Satellite Systems	6500.0	14330.05	GTO	geocentric	geostationary	Update
Amos-6	Satellite	No	Israel Aerospace Industries	5500.0	12100.0	GTO	geocentric	geostationary	Update
ArabSat 6A	Satellite	No	Lockheed Martin	6000.0	13227.74	GTO	geocentric	geostationary	Update
AsiaSat 6	Satellite	No	SSL	4428.0	9762.0	GTO	geocentric	geostationary	Update
AsiaSat 8	Satellite	No	SSL	4535.0	9998.0	GTO	geocentric	geostationary	Update
BRICSat 2	Satellite	No	US Naval Academy	1.0	2.2	LEO	geocentric	low-earth	Update
Bangabandhu-1	Satellite	No	Thales Alenia Space	3750.0	8270.0	GTO	geocentric	geostationary	Up
Beresheet	Lander	No	SSL	585.0	1289.7	GTO	geocentric	highly-elliptical	

Figure 32: Payloads Page

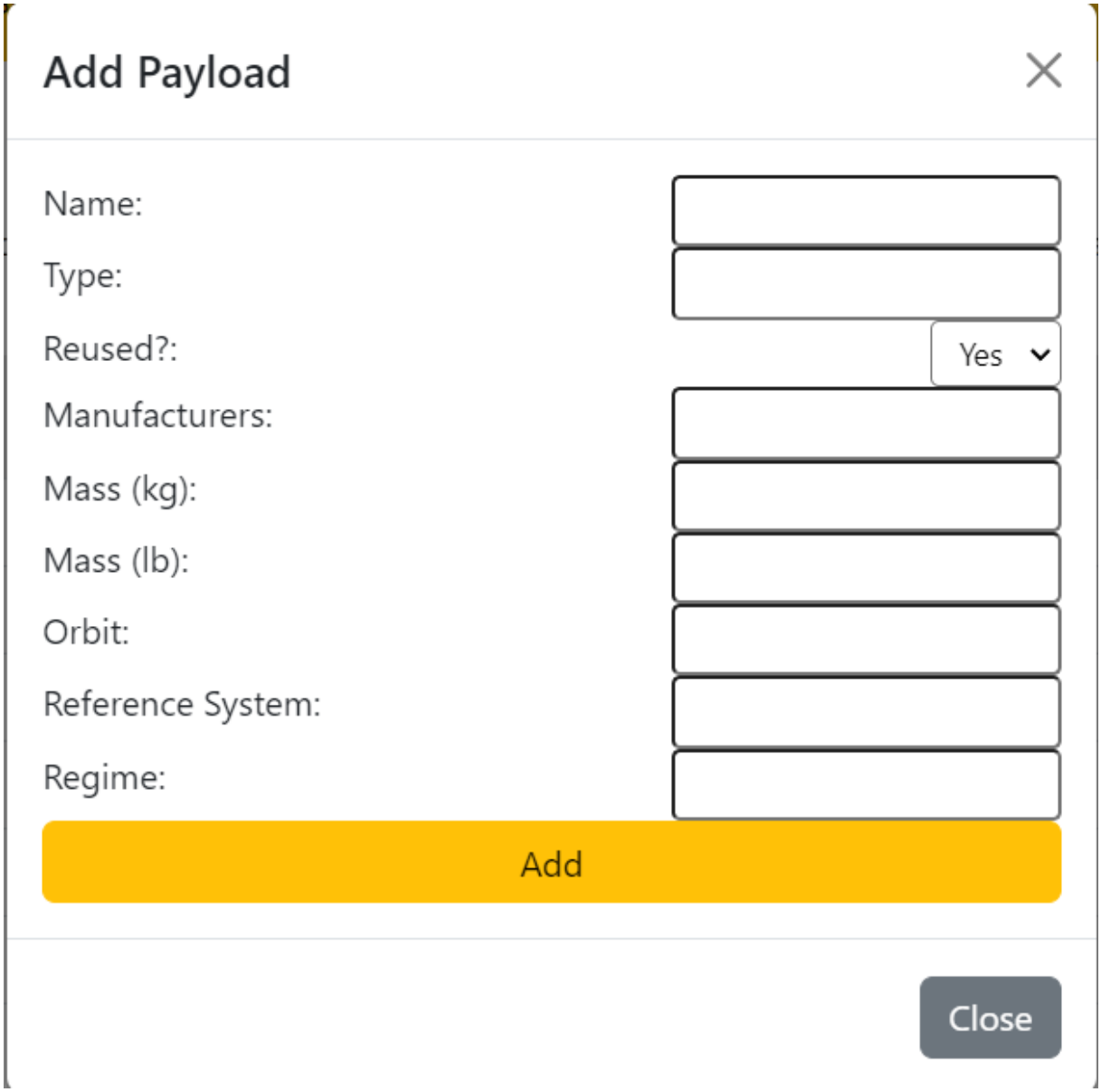


Figure 33: Add Payload Form

Name	Type	Is Reused	Manufacturers	Mass (kg)	Mass (lb)	Orbit	Reference System	Regime	Editing	
ABS-2A	Satellite	No	Boeing	1800.0	3950.0	GTO	geocentric	geostationary	Update	Delete
ABS-3A	Satellite	No	Boeing	1954.0	4307.0	GTO	geocentric	geostationary	Update	Delete
ANASIS-II	Satellite	No	South Korea	-	-	GEO	geocentric	geosynchronous	Update	Delete

Figure 34: Delete Payload Button

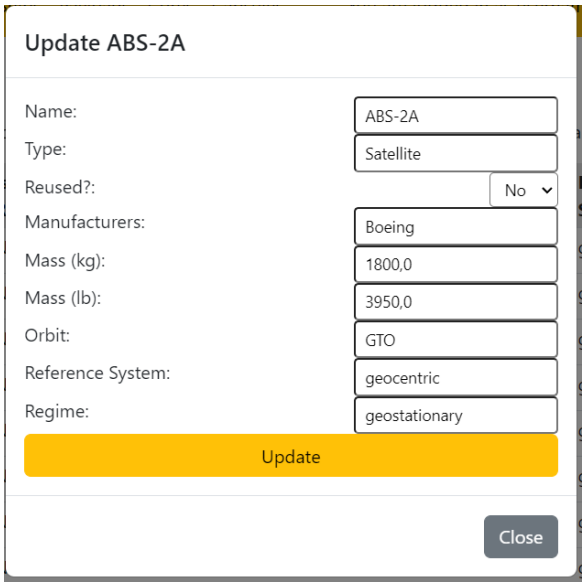


Figure 35: Update Payload Form

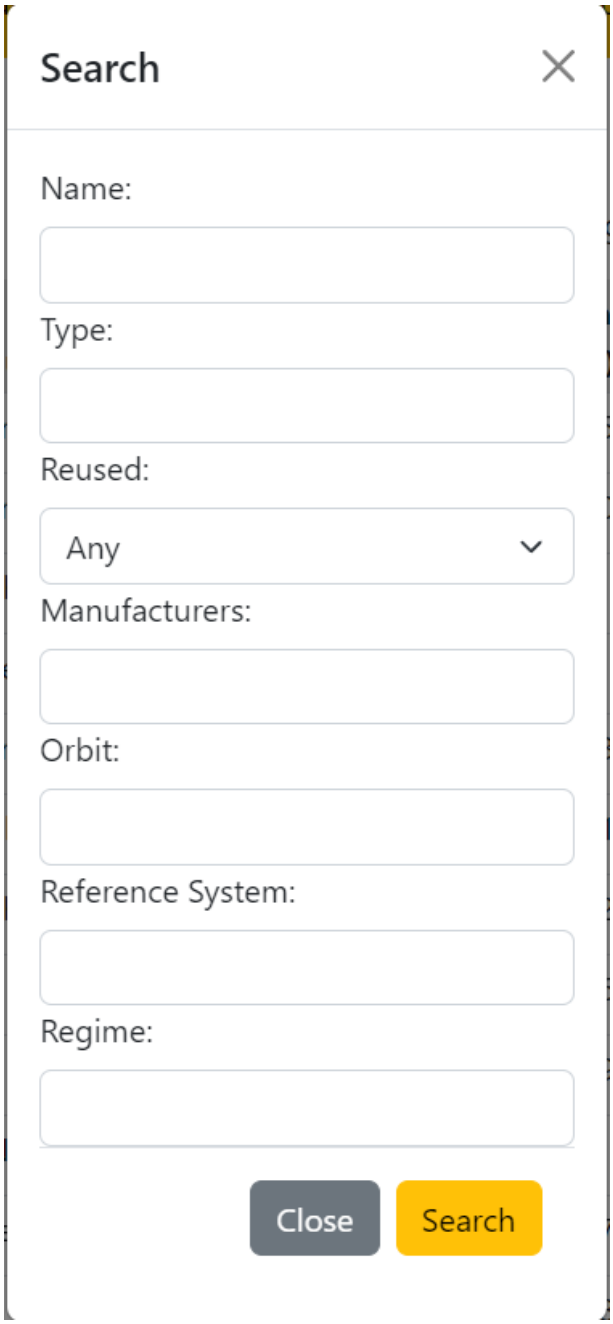


Figure 36: Search Payload Form

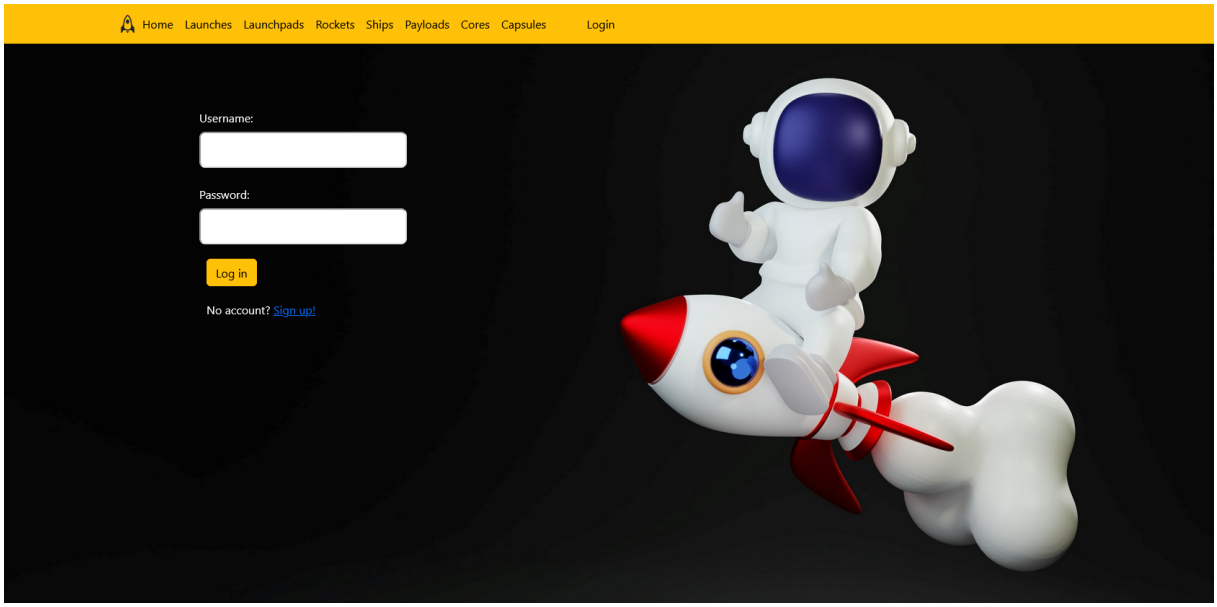


Figure 37: Login Page

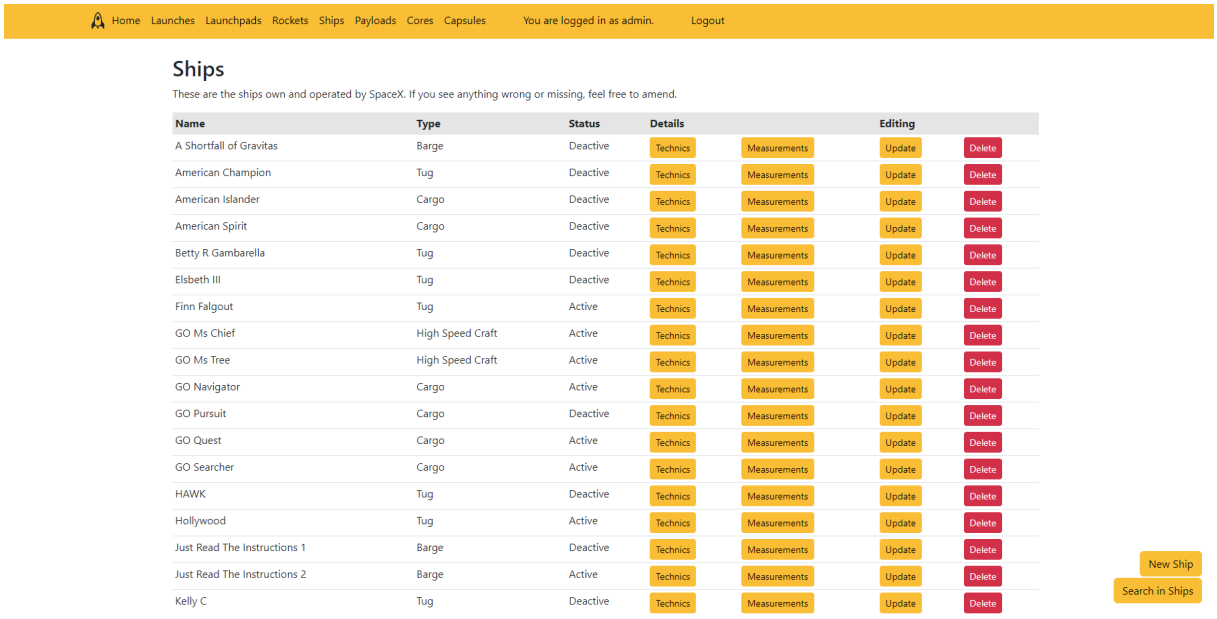


Figure 38: Ships Page

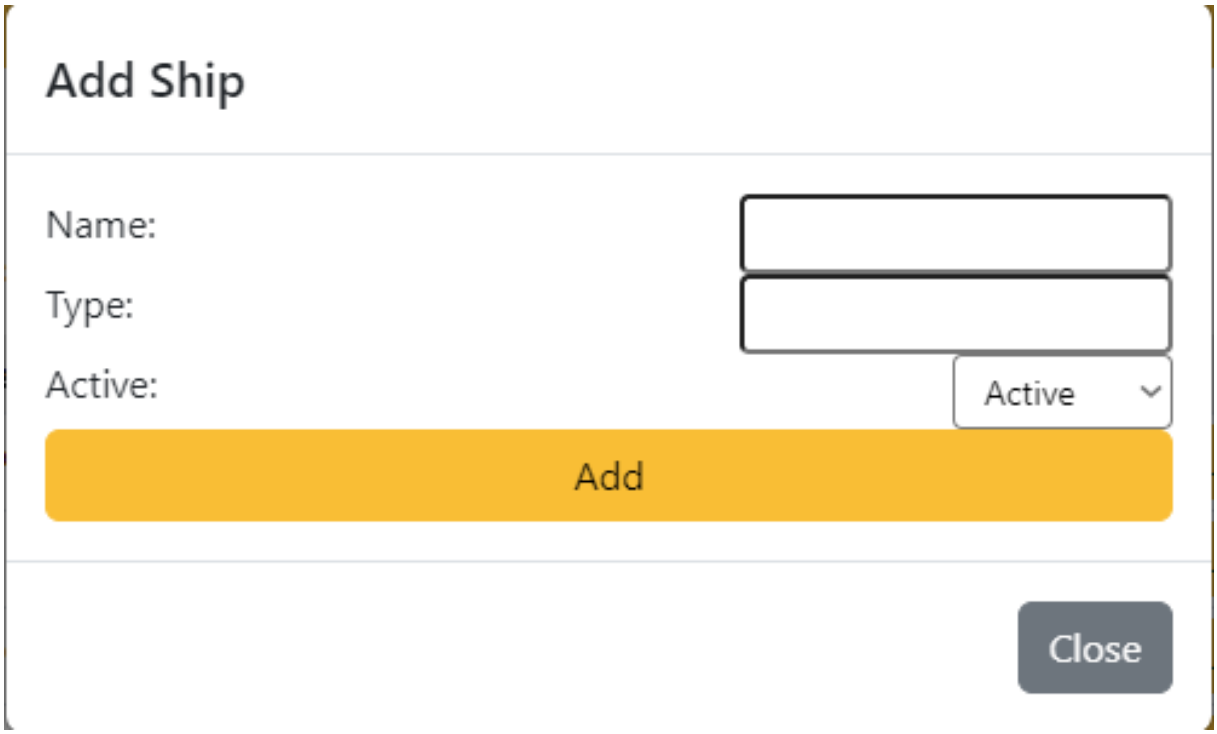


Figure 39: Add ship Form

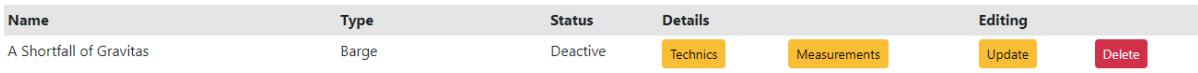


Figure 40: Delete ship Button

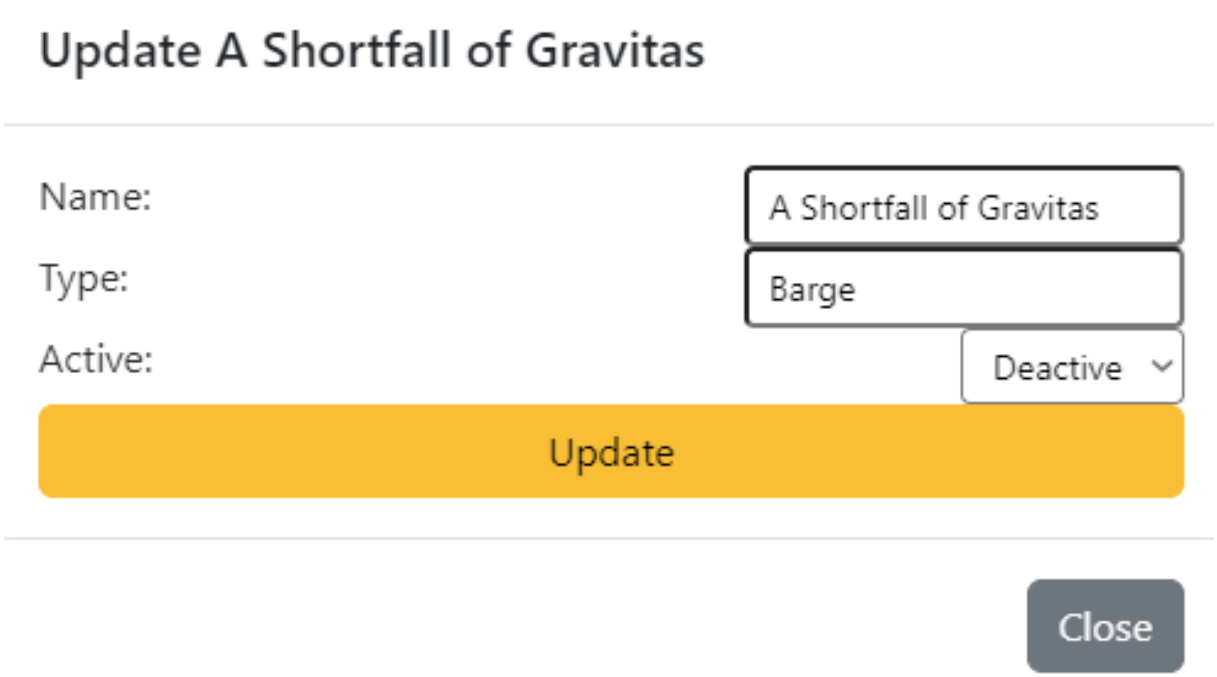


Figure 41: Update ship Form

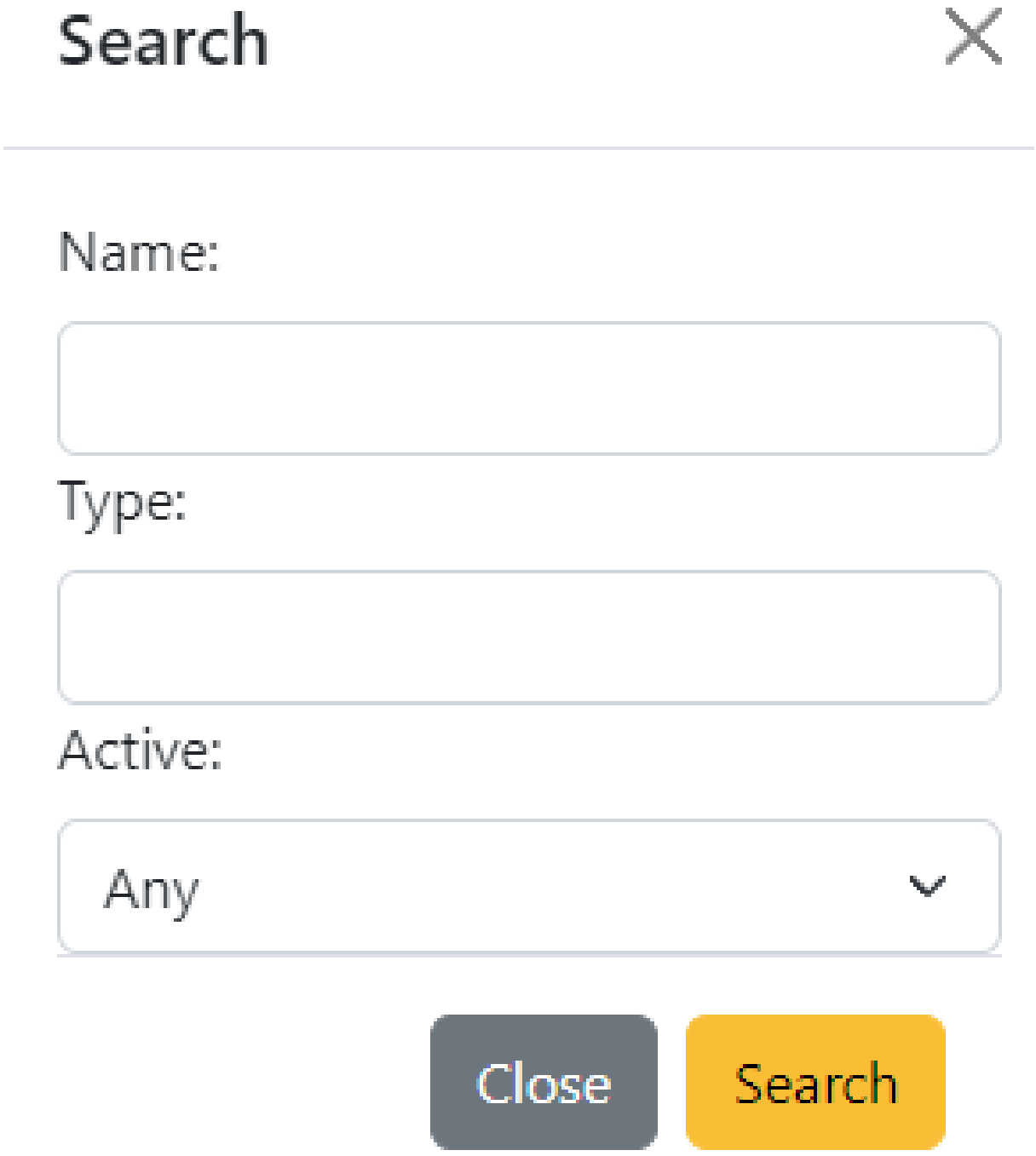


Figure 42: Search ship Form

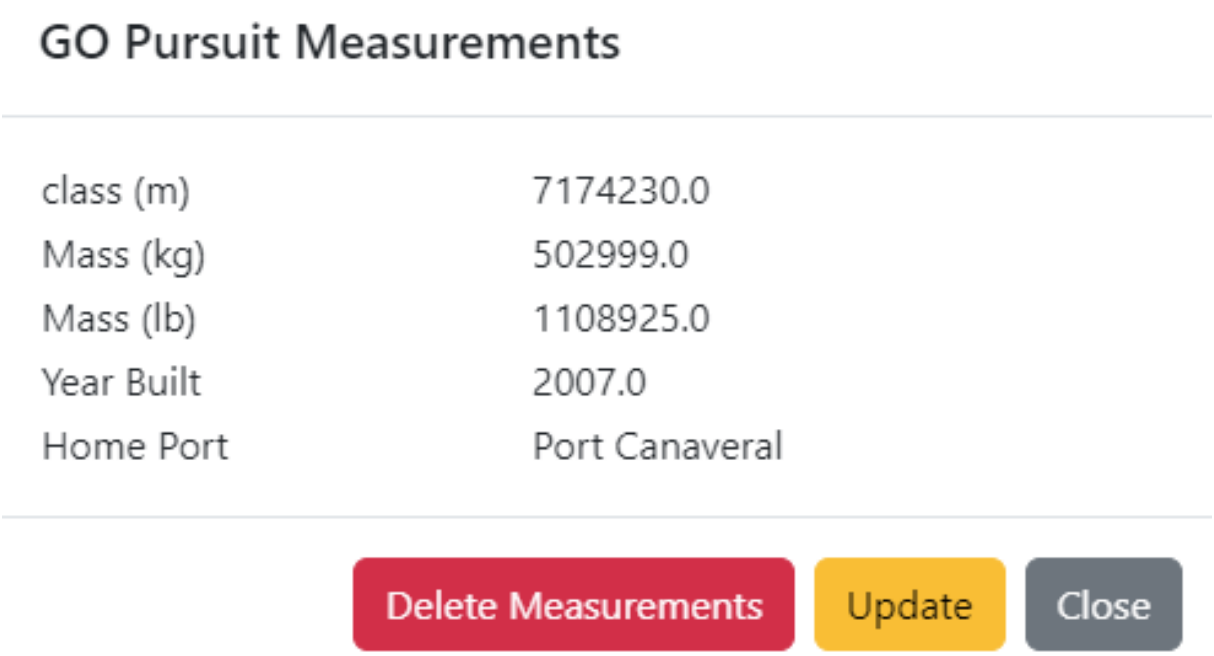


Figure 43: Measurement Modal

Figure 44: Add measurement Button

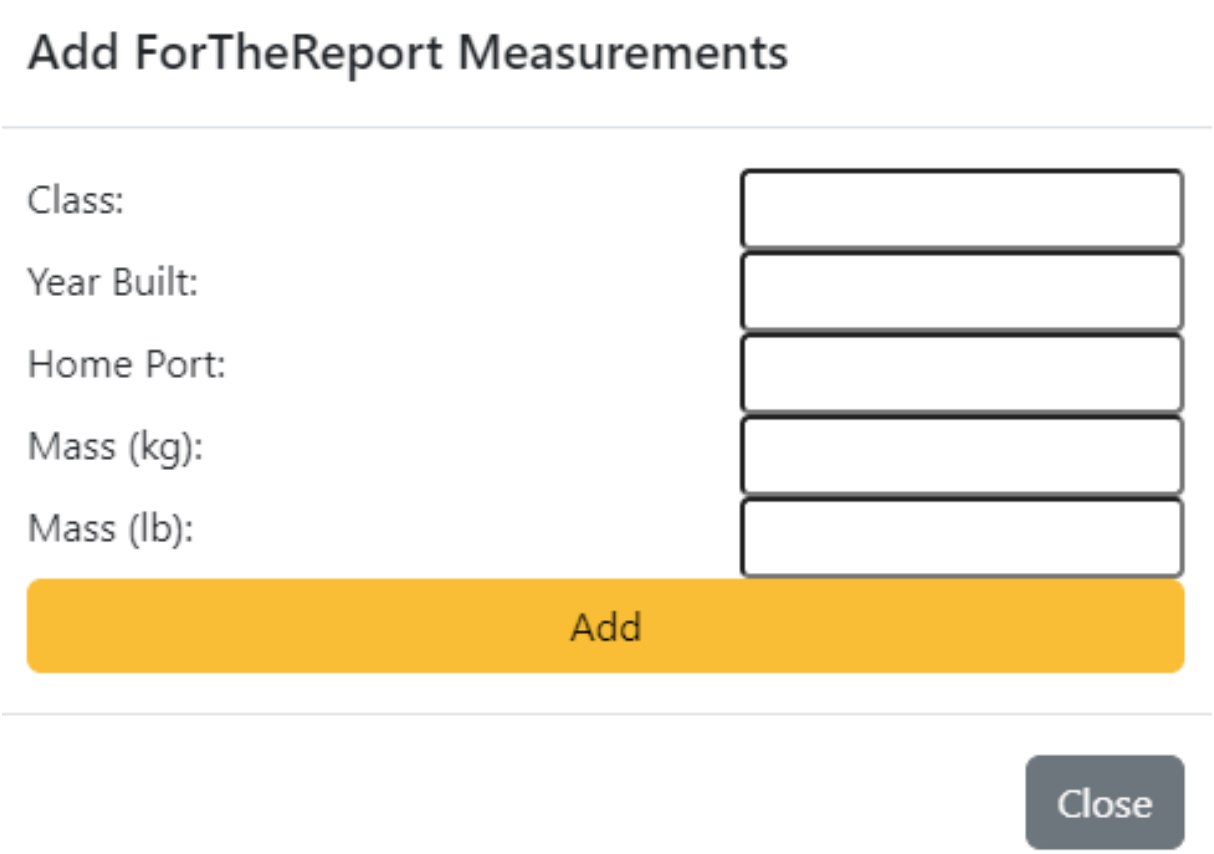


Figure 45: Add measurement Form

GO Pursuit Measurements

class (m)	7174230.0
Mass (kg)	502999.0
Mass (lb)	1108925.0
Year Built	2007.0
Home Port	Port Canaveral

Delete Measurements | Update | Close

Figure 46: Delete measurement Button

Update Betty R Gambarella Measurements

Class:	7427463,0
Mass (kg):	202302,0
Mass (lb):	446000,0
Year Built:	1974,0
Home Port:	Port of Los Angeles

Update

Close

Figure 47: Update Measurement Form