

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

CONTENTS

1	USER GUIDE	2
1.1	Parts Owned by Hakkı Arda Çoha	3
1.1.1	Capsules Page	3
1.1.2	Launches Page	3
1.1.3	Launch Details Modal	4
1.1.4	Add Launch	5
1.1.5	Delete Launch	5
1.1.6	Update Launch	6
1.1.7	Filter Launches	7
1.2	Parts Owned by Beyza Aydeniz	8
1.2.1	Engine Information Page	8
1.2.2	Add Engine Information	9
1.2.3	Delete Engine Information	9
1.2.4	Update Engine Information	10
1.2.5	Cores Page	11
1.2.6	Add Core	11
1.2.7	Delete Core	12
1.2.8	Update Core	12
1.2.9	Filter Core	13
1.3	Parts Owned by Abdullah Asım Emül	14
1.3.1	Rockets (Main)	14
	Rockets (Main) Viewing Page	14
	Rockets (Main) Adding Modal	14
	Rockets (Main) Editing Modal	14
	Rocket (Main) Deletion	14
1.3.2	Rocket Measurements	14
	Rocket Measurements Viewing Page	14
	Rocket Measurements Adding Modal	15
	Rocket Measurements Editing Modal	15
	Rocket Measurements Deletion	15
1.3.3	Rocket Images	15
	Rocket Image Viewing Page	15
	Rocket Image Adding Modal	17
	Rocket Image Deletion	17
1.3.4	Launchpads Viewing Page	17
1.3.5	Signup Page	17
1.4	Parts Owned by Alp Türkbayrak	18
1.4.1	Base HTML	18
1.4.2	Home Page	18
1.4.3	Ship Technics	18
1.4.4	Payloads Information Page	19
	Add Payload:	19
	Delete Payload:	19
	Update Payload:	19
	Search Payloads:	19
1.5	Parts Owned by Ahmet Metehan Yaman	20
1.5.1	User Login Page	20
1.5.2	Ships Page	20

	Add Ship:	20
	Delete Ships:	20
	Update ship:	20
	Search ships:	20
1.5.3	Ships Measurements	21
	Add measurement:	21
	Delete measurements:	21
	Update measurements:	21
2	DEVELOPER GUIDE	22
2.1	Parts Owned by Hakkı Arda Çoha	22
2.1.1	Launches Page	22
2.1.2	Capsules Page	23
2.1.3	Launch Adding	23
2.1.4	Launch Deleting	24
2.1.5	Launch Updating	24
2.1.6	Launch Filtering	25
2.1.7	Launch Details Adding	26
2.1.8	Launch Details Deleting	27
2.1.9	Launch Details Updating	27
2.1.10	Form Fields	28
2.2	Parts Owned by Beyza Aydeniz	29
2.2.1	Rocket Details 1 Table	29
2.2.2	Engine Information Page	29
2.2.3	Engine Information Deleting	30
2.2.4	Engine Information Adding	30
2.2.5	Engine Information Updating	31
2.2.6	Cores Table	32
2.2.7	Cores Page	32
2.2.8	Core Deleting	34
2.2.9	Core Adding	35
2.2.10	Core Updating	36
2.2.11	Core Filtering	38
2.3	Parts Owned by Abdullah Asım Emül	42
2.3.1	Code Structure	42
2.3.2	User Management	42
	Login	42
	Signup	43
2.3.3	Rockets (Main)	45
	Rockets (Main) Viewing Page	45
	Rockets (Main) Adding Modal	47
	Rockets (Main) Editing Modal	49
	Rocket (Main) Deletion	51
2.3.4	Rocket Measurements	52
	Rocket Measurements Viewing Page	52
	Rocket Measurements Adding Modal	54
	Rocket Measurements Editing Modal	56
	Rocket Measurements Deletion	58
2.3.5	Rocket Images	58
	Rocket Image Viewing Page	58
	Rocket Image Adding Modal	59

	Rocket Image Deletion	61
2.3.6	Launchpads Viewing Page	61
2.4	Parts Owned by Alp Türkbayrak	63
2.4.1	Base.html Page	63
	Bootstrap 5:	63
	Navbar:	63
2.4.2	Home.html Page	64
2.4.3	Ship Technics	65
	Adding Ship Technics:	66
	Updating Ship Technics:	66
	Deleting Ship Technics:	67
2.4.4	Payloads Page	67
	Displaying Payloads:	67
	Adding Payloads:	69
	Updating Payloads:	71
	Deleting Payloads:	74
	Searching Payloads:	74
2.5	Parts Owned by Ahmet Metehan Yaman	78
2.5.1	Ships Page Displaying	78
2.5.2	Ship Adding	80
2.5.3	Ship Deleting	81
2.5.4	Ships Updating	82
2.5.5	Ships Filtering	83
2.5.6	Ship Measurement Adding	84
2.5.7	Ship Measurement Updating	85
2.5.8	Ship Measurement Deleting	86

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.

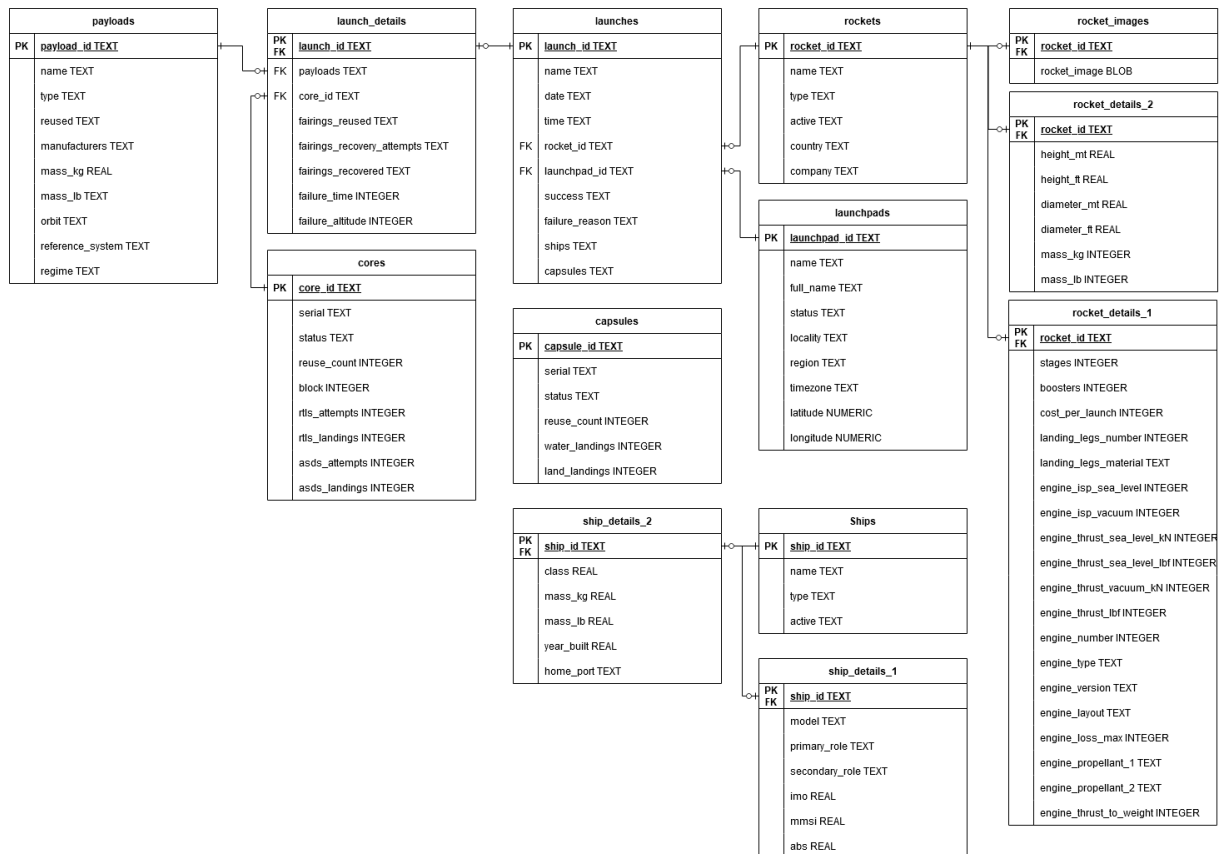
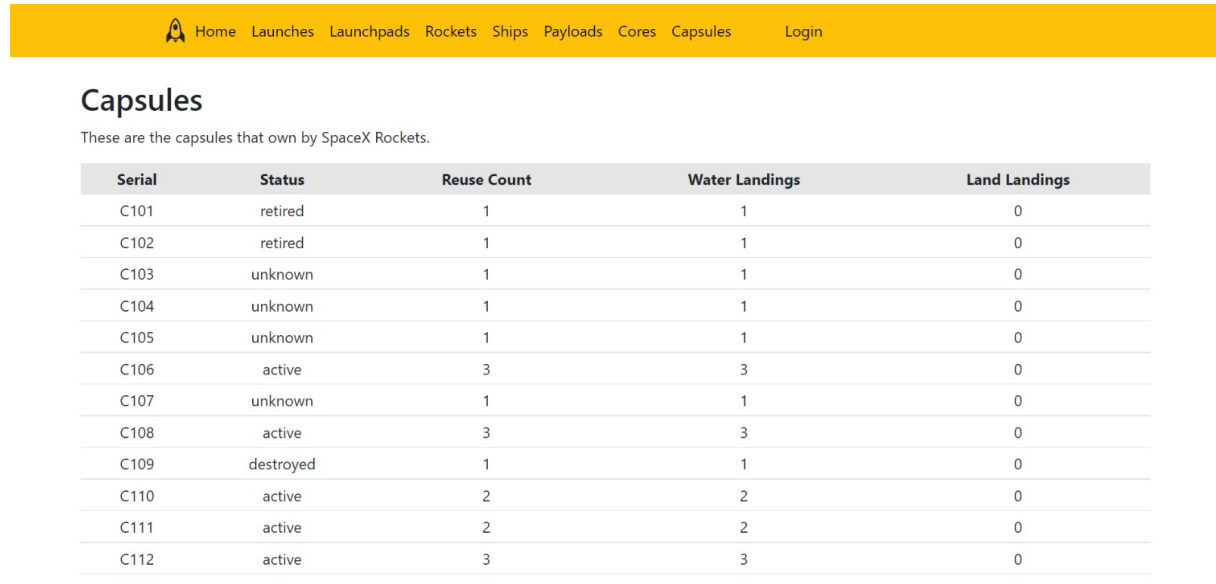


Figure 1: The ER Diagram of the SpaceX Database

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.


<div>  Home Launches Launchpads Rockets Ships Payloads Cores Capsules Login </div>										
<h2>Launches</h2> <p>These are the launches that have been conducted by SpaceX so far. If you see anything wrong or missing, feel free to amend.</p>										
Name	Date	Time	Rocket Name	Launchpad Name	Success	Failure Reason	Ship	Capsules	Editing	
FalconSat	2006-03-24	22:30	Falcon 1	Kwajalein Atoll	No	merlin engine failure			Details	Update
DemoSat	2007-03-21	01:10	Falcon 1	Kwajalein Atoll	No	harmonic oscillation leading to premature engine shutdown			Details	Update
Trailblazer	2008-08-03	03:34	Falcon 1	Kwajalein Atoll	No	residual stage-1 thrust led to collision between stage 1 and stage 2			Details	Update
RatSat	2008-09-28	23:15	Falcon 1	Kwajalein Atoll	Yes				Details	Update
RazakSat	2009-07-13	03:35	Falcon 1	Kwajalein Atoll	Yes				Details	Update
Falcon 9 Test Flight	2010-06-04	18:45	Falcon 9	CCSFS SLC 40	Yes				Details	New Launch
COTS 1	2010-12-08	15:43	Falcon 9	CCSFS SLC 40	Yes		American Champion	C101	Details	Update

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.

FalconSat Details

Payload ID

5eb0e4b5b6c3bb0006eeb1e1

Core ID

5e9e289df35918033d3b2623

Fairings Reused?

False

Fairings Recovery Attempt?

False

Fairings Recovered?

False

Failure Time

33

Failure Altitude

None

Delete Details

Update

Close

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.

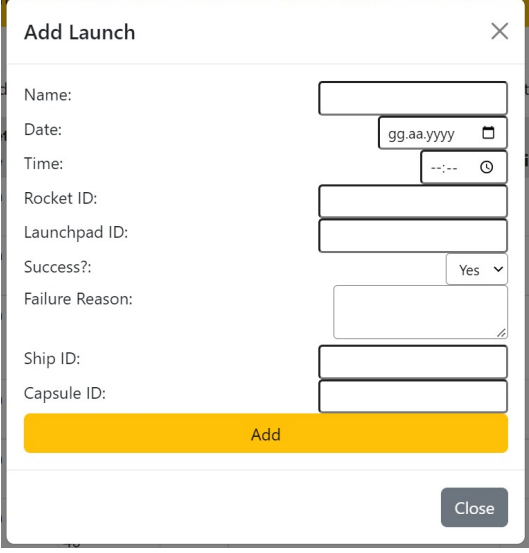
A screenshot of a web form titled "Add Launch" with a close button (X) in the top right corner. The form contains several input fields: "Name:" (text), "Date:" (calendar icon, pre-filled with "gg.aa.yyyy"), "Time:" (clock icon, pre-filled with "--:--"), "Rocket ID:" (text), "Launchpad ID:" (text), "Success?:" (dropdown menu with "Yes" selected), "Failure Reason:" (text area with a clear icon), "Ship ID:" (text), and "Capsule ID:" (text). At the bottom, there is a large yellow "Add" button and a grey "Close" button.

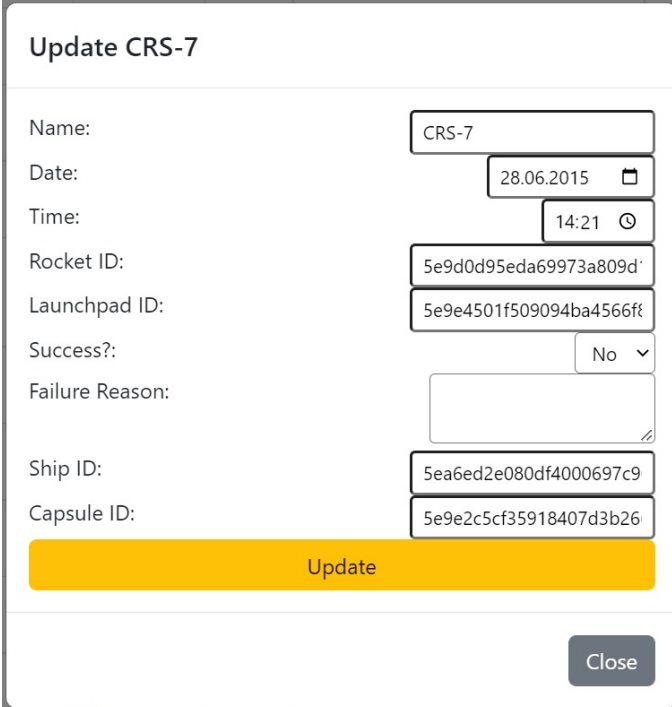
Figure 5: Adding data to Launches database

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.



The image shows a modal form titled "Update CRS-7". It contains several input fields for updating launch data. The fields are: Name (text input with "CRS-7"), Date (date picker with "28.06.2015"), Time (time picker with "14:21"), Rocket ID (text input with "5e9d0d95eda69973a809d"), Launchpad ID (text input with "5e9e4501f509094ba4566f"), Success? (dropdown menu with "No" selected), Failure Reason (text area), Ship ID (text input with "5ea6ed2e080df4000697c9"), and Capsule ID (text input with "5e9e2c5cf35918407d3b26"). At the bottom of the form is a large yellow "Update" button. In the bottom right corner of the modal is a grey "Close" button.

Name:	CRS-7
Date:	28.06.2015
Time:	14:21
Rocket ID:	5e9d0d95eda69973a809d
Launchpad ID:	5e9e4501f509094ba4566f
Success?:	No
Failure Reason:	
Ship ID:	5ea6ed2e080df4000697c9
Capsule ID:	5e9e2c5cf35918407d3b26

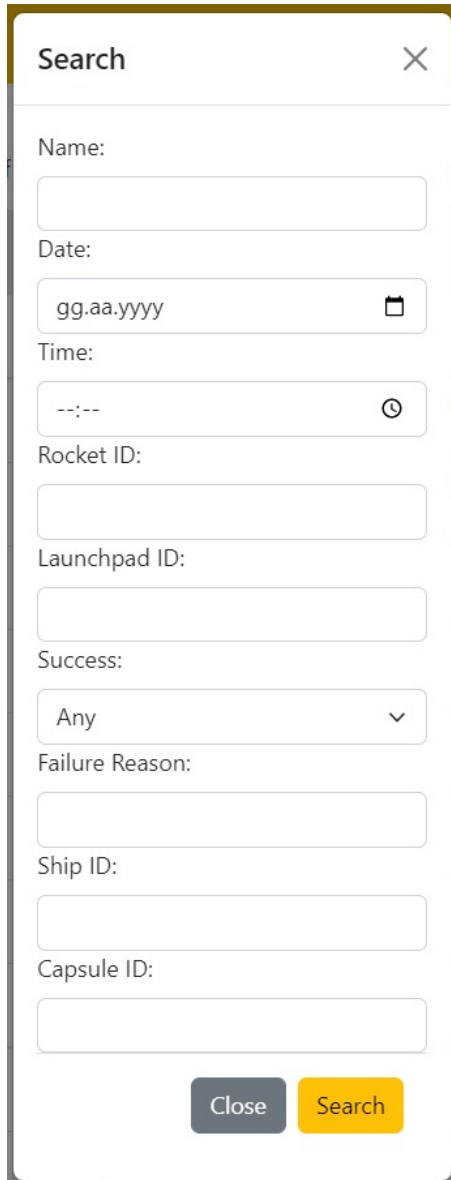
Update

Close

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.

A mobile application form titled "Search" with a close button (X) in the top right corner. The form contains several input fields: "Name:" (text), "Date:" (calendar icon, placeholder "gg.aa.yyyy"), "Time:" (clock icon, placeholder "--:--"), "Rocket ID:" (text), "Launchpad ID:" (text), "Success:" (dropdown menu with "Any" selected), "Failure Reason:" (text), "Ship ID:" (text), and "Capsule ID:" (text). At the bottom, there are two buttons: a gray "Close" button and a yellow "Search" button.

Search

Name:

Date:

Time:

Rocket ID:

Launchpad ID:

Success:

Failure Reason:

Ship ID:

Capsule ID:


Close Search

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.

 Home [Launches](#) [Launchpads](#) [Rockets](#) [Ships](#) [Payloads](#) [Cores](#) [Capsules](#) You are logged in as username. [Logout](#)

Rockets

These are the rockets own and operated by SpaceX. If you see anything wrong or missing, feel free to amend.

	Name	Type	Status	Country of Origin	Company	Details	Editing
	Falcon 1	rocket	Decommissioned	Republic of the Marshall Islands	SpaceX	Add Engine Info Measurements	Update Delete
	Falcon 9	rocket	Active	United States	SpaceX	Add Engine Info Measurements	Update Delete
	Falcon Heavy	rocket	Active	United States	SpaceX	Engine Info Measurements	Update Delete
	Starship	rocket	Decommissioned	United States	SpaceX	Engine Info Measurements	Update Delete
	Dumdum	splasher	Active	Middle of Nowhere	Xhibition	Add Engine Info Add Measurements	Update Delete
	Mars Rider	Flying Saucer	Decommissioned	Republic of Mars	Smace Inc.	Engine Info Measurements	Update Delete

[New Rocket](#)

Figure 8: Engine Information page entered from Rockets page

[Payloads](#) [Cores](#) [Capsules](#) You are logged in as username

Falcon 1 Engine Info

Stages	2
Boosters	0
Cost per launch	6700000
Landing legs number	0
Landing legs material	None
Engine Isp sea level	267
Engine Isp vacuum	304
Engine thrust sea level (kN)	420
Engine thrust sea level (lbf)	94000
Engine thrust vacuum (kN)	480
Engine thrust (lbf)	110000
Engine number	1
Engine type	Merlin
Engine version	1C
Engine layout	Single
Engine loss max	0
Engine propellant 1	Liquid Oxygen
Engine propellant 2	RP-1 Kerosene
Engine thrust to weight	96

[Delete Engine Info](#) [Update](#) [Close](#)

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.

Add Falcon 9 Engine Info

Stages:

Boosters:

Cost per launch:

Landing legs number:

Landing legs material:

Engine lsp sea level:

Engine lsp vacuum:

Engine thrust sea level (kN):

Engine thrust sea level (lbf):

Engine thrust vacuum (kN):

Engine thrust (lbf):

Engine number:

Engine type:

Engine version:

Engine layout:

Engine loss max:

Engine propellant 1:

Engine propellant 2:

Engine thrust to weight:

Add

Close

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.

Update Falcon 1 Engine Info	
Stages:	2
Boosters:	0
Cost per launch:	6700000
Landing legs number:	0
Landing legs material:	None
Engine lsp sea level:	267
Engine lsp vacuum:	304
Engine thrust sea level (kN):	420
Engine thrust sea level (lbf):	94000
Engine thrust vacuum (kN):	480
Engine thrust (lbf):	110000
Engine number:	1
Engine type:	Merlin
Engine version:	1C
Engine layout:	Single
Engine loss max:	0
Engine propellant 1:	Liquid Oxygen
Engine propellant 2:	RP-1 Kerosene
Engine thrust to weight:	96
<div>Update</div>	
<div>Close</div>	

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.

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	0	0	0	0	0	0	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
B0006	lost	0	1	0	0	0	0	Delete	Update
B0007	lost	0	1	0	0	0	0	Delete	Update
B1003	lost	0	1	0	0	0	0	Delete	Update
B1004	lost	0	1	0	0	0	0	Delete	New Core
B1005	lost	0	1	0	0	0	0	Delete	Filter 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.

Add Core

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:

Add

Close

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

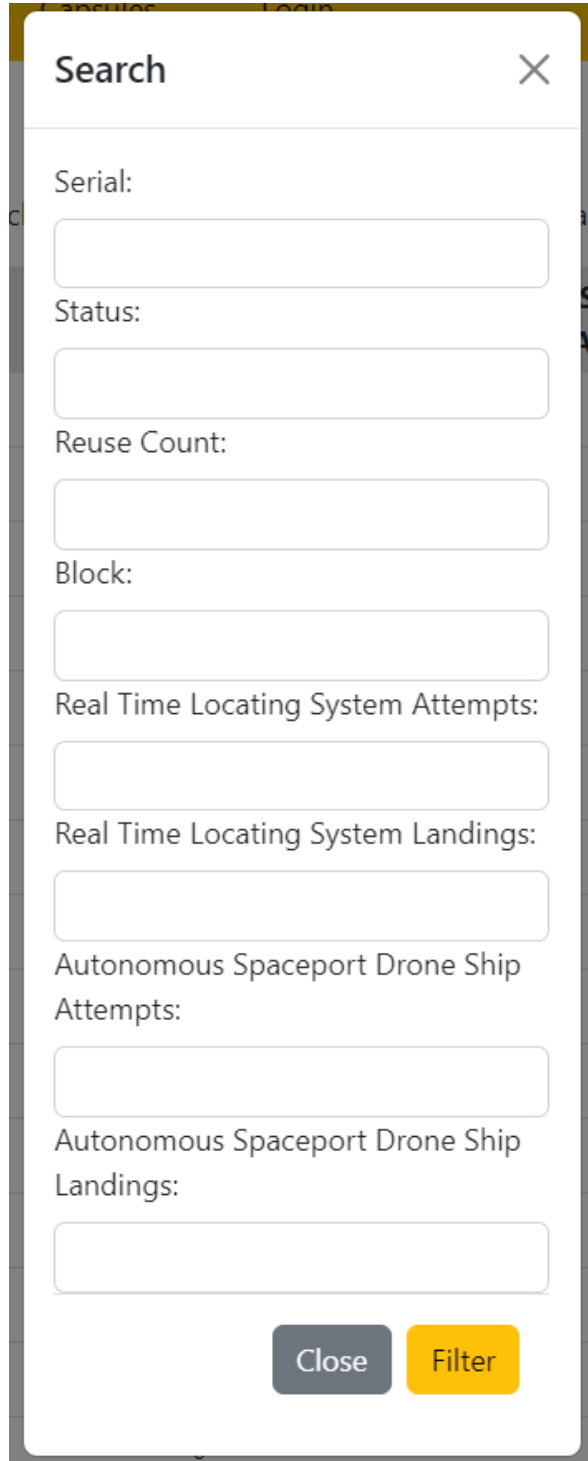
Update

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.




The image shows a mobile application interface with a modal window titled "Search" in the top left corner, accompanied by a close icon (X) in the top right. The modal contains a series of labels followed by empty text input fields: "Serial:", "Status:", "Reuse Count:", "Block:", "Real Time Locating System Attempts:", "Real Time Locating System Landings:", "Autonomous Spaceport Drone Ship Attempts:", and "Autonomous Spaceport Drone Ship Landings:". At the bottom of the modal, there are two buttons: a gray "Close" button and a yellow "Filter" button.

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)



Home

Launches

Launchpads

Rockets

Ships

Payloads

Cores









Capsules

You are logged in as 1234.

Logout

Rockets

These are the rockets own and operated by SpaceX. If you see anything wrong or missing, feel free to amend.

	Name	Type	Status	Country of Origin	Company	Details		Editing	
	Falcon 1	rocket	Decommissioned	Republic of the Marshall Islands	SpaceX	Engine Info	Measurements	Update	Delete
	Falcon 9	rocket	Active	United States	SpaceX	Engine Info	Measurements	Update	Delete
	Falcon Heavy	rocket	Active	United States	SpaceX	Engine Info	Measurements	Update	Delete
	Starship	rocket	Decommissioned	United States	SpaceX	Engine Info	Measurements	Update	Delete
	Dumdum	splasher	Active	Middle of Nowhere	Xhibition	Add Engine Info	Add Measurements	Update	Delete
	Mars Rider	Flying Saucer	Decommissioned	Republic of Mars	Smace Inc.	Engine Info	Measurements	Update	Delete
	tadfasdff		Active			Add Engine Info	Add Measurements	Update	Delete

New Rocket

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.

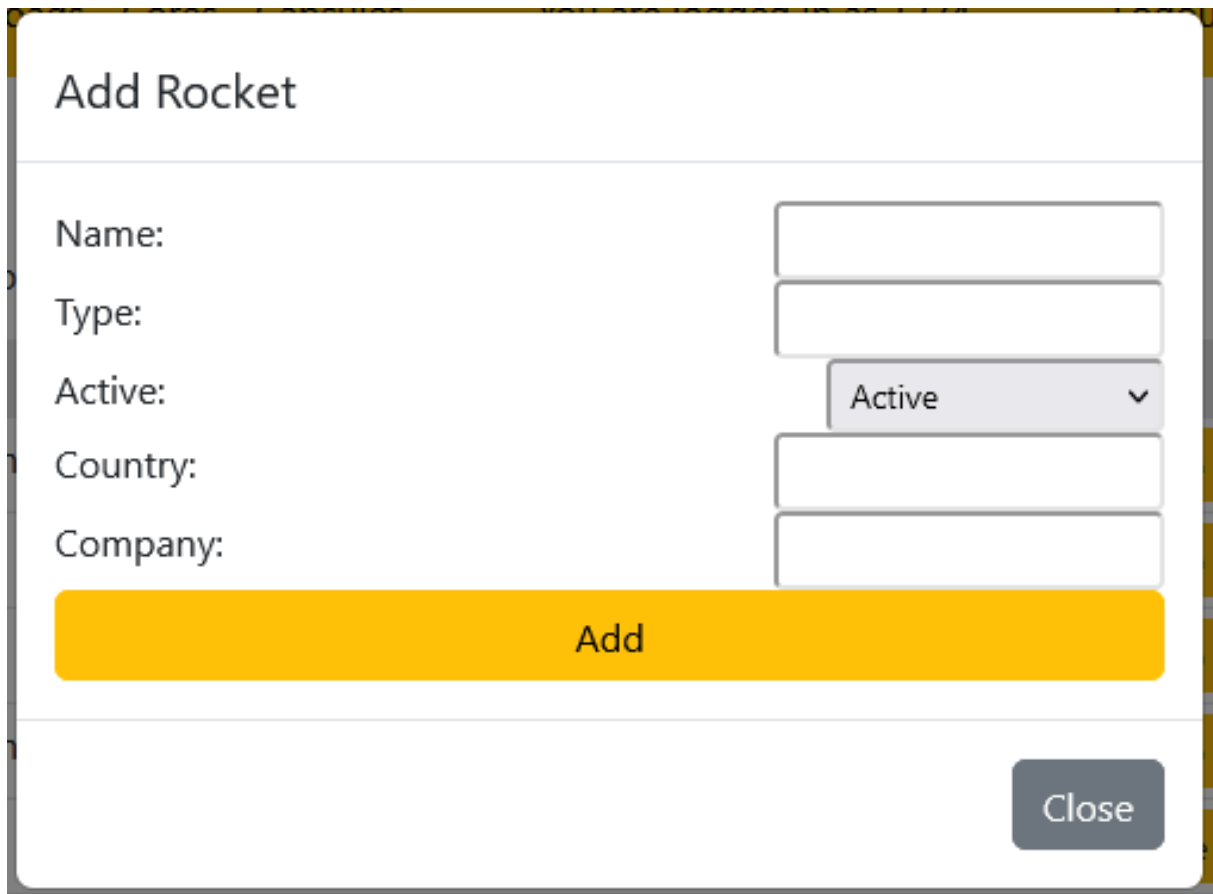
After clicking this button a modal will appear allowing the user to enter the information of the rocket they wish to add. This information is the same as the one displayed on the rockets viewing page. Once all desired information has been entered, the “Add” button will add the information to the system.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

A modal window titled "Add Rocket" with a white background and a thin grey border. It contains five input fields: "Name:", "Type:", "Active:", "Country:", and "Company:". The "Active:" field is a dropdown menu with "Active" selected and a downward arrow. Below the fields is a large yellow button labeled "Add". In the bottom right corner is a grey button labeled "Close".

Add Rocket

Name:

Type:

Active: Active ▼

Country:

Company:

Add

Close

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)

Update Falcon 1

Name: Falcon 1

Type: rocket

Active: Decommissioned ▾

Country: Republic of the Marshal

Company: SpaceX

Update

Close

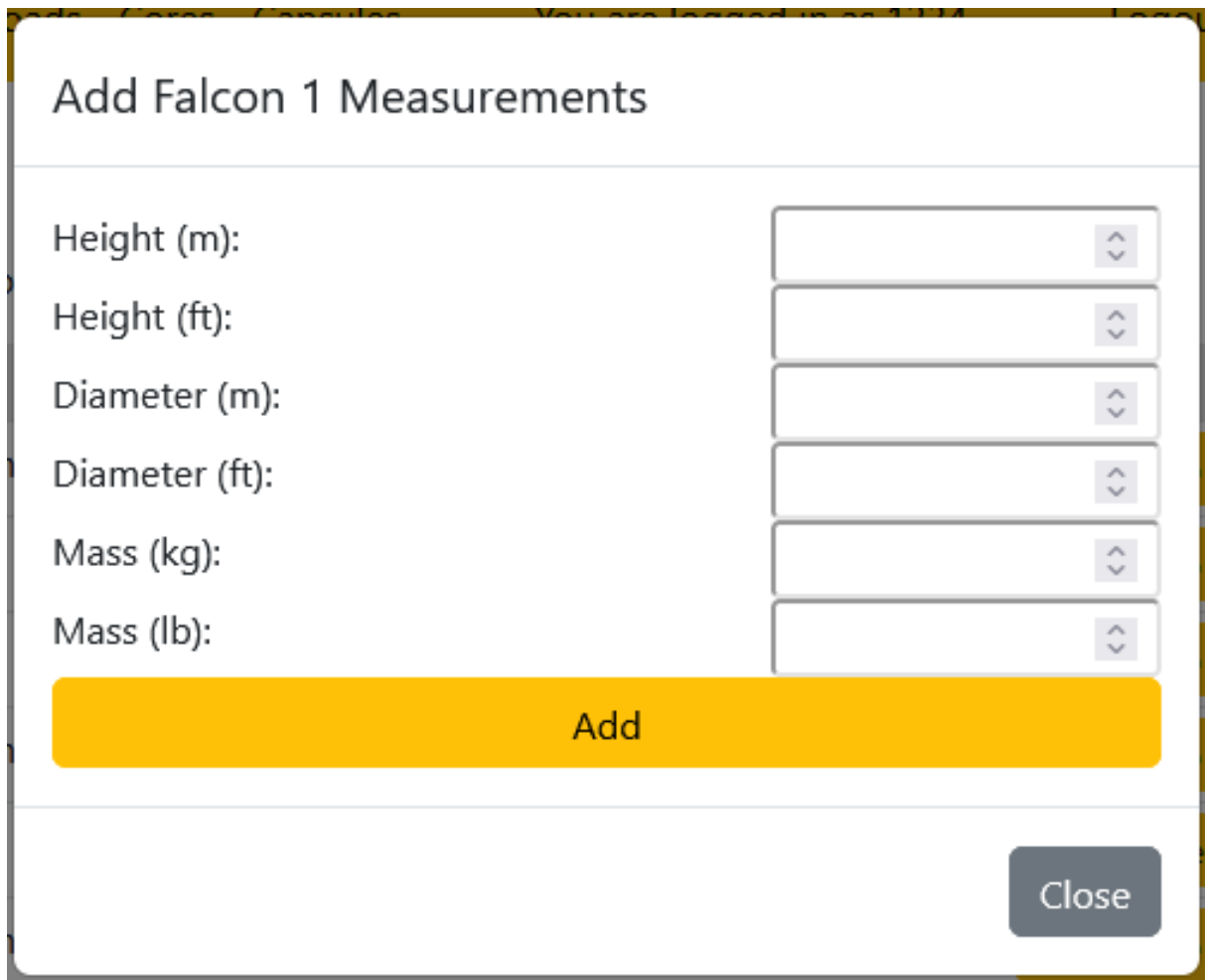
Figure 19: Main rockets editing view

Falcon 1 Measurements

Height (m)	22.25
Height (ft)	73.0
Diameter (m)	1.68
Diameter (ft)	5.5
Mass (kg)	30146
Mass (lb)	66460

Delete Measurements Update Close

Figure 20: Rocket measurements view

A modal window titled "Add Falcon 1 Measurements" with a white background and a thin grey border. It contains six input fields for measurements: Height (m), Height (ft), Diameter (m), Diameter (ft), Mass (kg), and Mass (lb). Each field is a white rectangle with a small grey arrow icon on the right. Below the fields is a large yellow button labeled "Add". In the bottom right corner is a grey button labeled "Close".

Add Falcon 1 Measurements

Height (m):

Height (ft):

Diameter (m):

Diameter (ft):

Mass (kg):

Mass (lb):

Add

Close

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

Update Falcon 1 Measurements

Height (m):	22.25
Height (ft):	73.0
Diameter (m):	1.68
Diameter (ft):	5.5
Mass (kg):	30146
Mass (lb):	66460

Update

Close

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:

```
1 @views.route('/launches', methods=['GET', 'POST'])
2 def launches():
3     launch_data = database.get_launches()
4     launch_details = database.get_launch_details()
5
6     inexistent_launch_detail = []
7     inexistent_launch_detail_dict = []
8
9     rocket_data = database.get_rockets()
10    rocket_dict = {}
11    for rocket in rocket_data:
12        rocket_dict[rocket["rocket_id"]] = rocket["name"].replace(" ", " ")
13
14    launchpad_data = database.get_launchpads()
15    launchpad_dict = {}
16    for launchpad in launchpad_data:
17        launchpad_dict[launchpad["launchpad_id"]] = launchpad["name"]
18
19    ship_data = database.get_ships_only()
20    ship_dict = {}
21    for ship in ship_data:
22        ship_dict[ship["ship_id"]] = ship["name"]
23
24    capsule_data = database.get_capsules()
25    capsule_dict = {}
26    for capsule in capsule_data:
27        capsule_dict[capsule["capsule_id"]] = capsule["serial"]
28
29    present = False
30    for launch in launch_data:
31        for launch_detail in launch_details:
32            if launch["launch_id"] == launch_detail["launch_id"]:
33                present = True
34                break
35        if not present:
36            inexistent_launch_detail += [launch["launch_id"]]
37            inexistent_launch_detail_dict += [{"launch_id": launch["
launch_id"], "name": launch["name"]}]]
38        present = False
39
40    launch_data = database.get_launches()
41    return render_template("launches.html", launches=launch_data,
launch_details = launch_details,
42    inexistent_launch_detail=inexistent_launch_detail,
inexistent_launch_detail_dict=inexistent_launch_detail_dict,
43    formM=forms.LaunchForm(), formD=forms.LaunchDetailForm(),
44    rockets=rocket_dict, launchpads = launchpad_dict, ships = ship_dict,
capsules = capsule_dict)
```

SQL statement in database.py:

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

```

3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             "SELECT * FROM launches LEFT JOIN launch_details on launches.
7             launch_id = launch_details.launch_id"
            ).fetchall()

1 def get_launch_details():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             """SELECT * FROM launch_details
7             JOIN (SELECT launch_id, name FROM launches) AS launch_names
8             ON launch_details.launch_id = launch_names.launch_id"""
9             ).fetchall()

```

2.1.2 Capsules Page

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

```

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

```

SQL statement in database.py:

```

1 def get_capsules():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             "SELECT * FROM capsules"
7             ).fetchall()

```

2.1.3 Launch Adding

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

```

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

```

SQL statement in database.py:

```

1 def add_launch(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new launch
7         launch_columns = cur.execute("PRAGMA table_info(launches)").
            fetchall()
8
9         new_launch = [str(current_app.config["launch_id"])]
10        current_app.config["launch_id"] += 1
11        for column in launch_columns:
12            if column["name"] in request.form.keys():
13                new_launch += [request.form[column["name"]]]

```

```

14
15         cur.execute(f'INSERT INTO launches VALUES ({",".join("? " * len(
launch_columns))})', new_launch)
16         con.commit()

```

2.1.4 Launch Deleting

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

```

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

```

SQL statement in database.py:

```

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

```

2.1.5 Launch Updating

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

```

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

```

SQL statement in database.py:

```

1 def update_launch(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Update launch
7         launch_columns = cur.execute("PRAGMA table_info(launches)").
fetchall()
8         launch_columns_str = ",".join([column["name"] + "=? " for column in
launch_columns if column["name"] != "launch_id"])
9
10        launch = []
11        for column in launch_columns:
12            if column["name"] in request.form.keys():
13                launch += [request.form[column["name"]]]
14
15        launch_id = launch[0]
16        launch = launch[1:] + [launch_id]
17
18        cur.execute(f'UPDATE launches SET {launch_columns_str} WHERE
launch_id = ?', launch)
19        con.commit()

```

2.1.6 Launch Filtering

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

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

SQL statement in database.py:

```
1 def filter_launches(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         print("filter_launches executed")
6         query = "SELECT * FROM launches"
7         params = []
8         print(request.form.get('fname'))
9         if request.form.get('fname'):
10             if not params:
11                 query += " WHERE"
12                 user_name = request.form['fname']
13                 query += " name LIKE ?"
14                 param_name = "%" + user_name + "%"
15                 params.append(param_name)
16
17         if request.form.get('fdate'):
18             if not params:
19                 query += " WHERE"
20             else:
21                 query += " AND"
22                 user_date = request.form['fdate']
23                 query += " date LIKE ?"
24                 param_date = "%" + user_date + "%"
25                 params.append(param_date)
26
27         if request.form.get('ftime') != "":
28             if not params:
29                 query += " WHERE"
30             else:
31                 query += " AND"
32                 user_time = request.form['ftime']
33                 query += " time LIKE ?"
34                 param_time = "%" + user_time + "%"
35                 params.append(param_time)
36
37         if request.form.get('frocket_id'):
38             if not params:
39                 query += " WHERE"
40             else:
41                 query += " AND"
42                 user_rocket_id = request.form['frocket_id']
43                 query += " rocket_id LIKE ?"
44                 param_rocket_id = "%" + user_rocket_id + "%"
45                 params.append(param_rocket_id)
46
47         if request.form.get('flaunchpad_id'):
48             if not params:
49                 query += " WHERE"
50             else:
```

```

51         query += " AND"
52         user_launchpad_id = request.form['flaunchpad_id']
53         query += " launchpad_id LIKE ?"
54         param_launchpad_id = "%" + user_launchpad_id + "%"
55         params.append(param_launchpad_id)
56
57     if request.form.get('fsuccess'):
58         if not params:
59             query += " WHERE"
60         else:
61             query += " AND"
62             user_success = request.form['fsuccess']
63             query += " success LIKE ?"
64             param_success = "%" + user_success + "%"
65             params.append(param_success)
66
67     if request.form.get('ffailure_reason'):
68         if not params:
69             query += " WHERE"
70         else:
71             query += " AND"
72             user_failure_reason = request.form['ffailure_reason']
73             query += " failure_reason LIKE ?"
74             param_failure_reason = "%" + user_failure_reason + "%"
75             params.append(param_failure_reason)
76
77     if request.form.get('fship'):
78         if not params:
79             query += " WHERE"
80         else:
81             query += " AND"
82             user_ship = request.form['fship']
83             query += " ship LIKE ?"
84             param_ship = "%" + user_ship + "%"
85             params.append(param_ship)
86
87     if request.form.get('fcapsules'):
88         if not params:
89             query += " WHERE"
90         else:
91             query += " AND"
92             user_capsules = request.form['fcapsules']
93             query += " capsules LIKE ?"
94             param_capsules = "%" + user_capsules + "%"
95             params.append(param_capsules)
96
97     print(query)
98     print(tuple(params))
99     filter = cur.execute(query, tuple(params)).fetchall()
100    return filter

```

2.1.7 Launch Details Adding

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

```

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

```

SQL statement in database.py:

```

1 def add_launch_details(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new launch_details
7         launch_details_columns = cur.execute("PRAGMA table_info(
launch_details)").fetchall()
8
9         new_launch_details = []
10        for column in launch_details_columns:
11            if column["name"] in request.form.keys():
12                new_launch_details += [request.form[column["name"]]]
13
14        cur.execute(f'INSERT INTO launch_details VALUES ({",".join("? " *
len(launch_details_columns))})', new_launch_details)
15        con.commit()

```

2.1.8 Launch Details Deleting

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

```

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

```

SQL statement in database.py:

```

1 def delete_launch_details(launch_id):
2     with sqlite3.connect(db_location) as con:
3         cursor = con.cursor()
4         query = "DELETE FROM launch_details WHERE (launch_id = ?)"
5         cursor.execute(query, (launch_id,))
6         con.commit()

```

2.1.9 Launch Details Updating

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

```

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

```

SQL statement in database.py:

```

1 def update_launch_detail(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Update launch_detail
7         launch_detail_columns = cur.execute("PRAGMA table_info(
launch_details)").fetchall()
8         launch_detail_str = ",".join([column["name"] + "=? " for column in
launch_detail_columns if column["name"] != "launch_id"])
9
10        launch_detail = []
11        for column in launch_detail_columns:
12            if column["name"] in request.form.keys():

```

```

13         launch_detail += [request.form[column["name"]]]
14
15         launch_id = launch_detail[0]
16         launch_detail = launch_detail[1:] + [launch_id]
17
18         cur.execute(f'UPDATE launch_details SET {launch_detail_str} WHERE
19 launch_id = ?', launch_detail)
20         con.commit()

```

2.1.10 Form Fields

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

```

1 class LaunchForm(FlaskForm):
2     launch_id = StringField()
3     name = StringField()
4     date = DateField()
5     time = TimeField()
6     rocket_id = StringField()
7     launchpad_id = StringField()
8     success = SelectField(choices=[("True", "Yes"), ("False", "No")])
9     failure_reason = TextAreaField()
10    ship = StringField()
11    capsules = StringField()

```

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

```

1 class LaunchDetailForm(FlaskForm):
2     launch_id = StringField()
3     payloads = StringField()
4     core_id = StringField()
5     fairings_reused = SelectField(choices=[("True", "Yes"), ("False", "No")
6 ])
7     fairings_recovery_attempts = SelectField(choices=[("True", "Yes"), ("
8 False", "No")])
9     fairings_recovered = SelectField(choices=[("True", "Yes"), ("False", "
10 No")])
11    failure_time = IntegerField()
12    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:

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

1     inexistent_rocket_d1 = []

1     inexistent_rocket_d1_dict = []

1     for rocket in rocket_data:
2         for rocket_d1 in rocket_d1_data:
3             if rocket["rocket_id"] == rocket_d1["rocket_id"]:
4                 present = True
5                 break
6             if not present:
7                 inexistent_rocket_d1 += [rocket["rocket_id"]]
8                 inexistent_rocket_d1_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]]
9                 present = False

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

SQL statement in database.py:

```
1 def get_rocket_d1():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             """SELECT * FROM rocket_details_1
7             JOIN (SELECT rocket_id, name FROM rockets) AS rocket_names
8             ON rocket_details_1.rocket_id = rocket_names.rocket_id"""
9         ).fetchall()
```


2.2.3 Engine Information Deleting

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

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

SQL statement for deleting engine information in database.py:

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

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

```
1 function delete_rocket_d1(rocket_id) {
2     document.location = '/delete_rocket_detail_1?rocket_id=' + rocket_id
3 }
```

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:

```
1 class RocketD1Form(FlaskForm):
2     rocket_id = StringField()
3     stages = IntegerField(validators=[validators.InputRequired()])
4     boosters = IntegerField(validators=[validators.InputRequired()])
5     cost_per_launch = IntegerField(validators=[validators.InputRequired()])
6     landing_legs_number = IntegerField(validators=[validators.InputRequired()])
7     landing_legs_material = StringField()
8     engine_isp_sea_level = IntegerField(validators=[validators.
9     InputRequired()])
10    engine_isp_vacuum = IntegerField(validators=[validators.InputRequired()])
11    engine_thrust_sea_level_kN = IntegerField(validators=[validators.
12    InputRequired()])
13    engine_thrust_sea_level_lbf = IntegerField(validators=[validators.
14    InputRequired()])
15    engine_thrust_vacuum_kN = IntegerField(validators=[validators.
16    InputRequired()])
17    engine_thrust_lbf = IntegerField(validators=[validators.InputRequired()])
18    engine_number = IntegerField(validators=[validators.InputRequired()])
19    engine_type = StringField()
20    engine_version = StringField()
21    engine_layout = StringField()
22    engine_loss_max = IntegerField(validators=[validators.InputRequired()])
23    engine_propellant_1 = StringField()
24    engine_propellant_2 = StringField()
25    engine_thrust_to_weight = DecimalField(validators=[validators.
26    InputRequired()])
```

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

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

```

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

```

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

```

1 def add_rocket_d1(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new rocket_d1
7         rocket_d1_columns = cur.execute("PRAGMA table_info(rocket_details_1
8         )").fetchall()
9
10        new_rocket_d1 = []
11        for column in rocket_d1_columns:
12            if column["name"] in request.form.keys():
13                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:

```

1 class RocketD1Form(FlaskForm):
2     rocket_id = StringField()
3     stages = IntegerField(validators=[validators.InputRequired()])
4     boosters = IntegerField(validators=[validators.InputRequired()])
5     cost_per_launch = IntegerField(validators=[validators.InputRequired()])
6     landing_legs_number = IntegerField(validators=[validators.InputRequired()])
7     landing_legs_material = StringField()
8     engine_isp_sea_level = IntegerField(validators=[validators.
9     InputRequired()])
10    engine_isp_vacuum = IntegerField(validators=[validators.InputRequired()])
11    engine_thrust_sea_level_kN = IntegerField(validators=[validators.
12    InputRequired()])
13    engine_thrust_sea_level_lbf = IntegerField(validators=[validators.
14    InputRequired()])
15    engine_thrust_vacuum_kN = IntegerField(validators=[validators.
16    InputRequired()])
17    engine_thrust_lbf = IntegerField(validators=[validators.InputRequired()])
18    engine_number = IntegerField(validators=[validators.InputRequired()])
19    engine_type = StringField()
20    engine_version = StringField()
21    engine_layout = StringField()
22    engine_loss_max = IntegerField(validators=[validators.InputRequired()])
23    engine_propellant_1 = StringField()
24    engine_propellant_2 = StringField()
25    engine_thrust_to_weight = DecimalField(validators=[validators.
26    InputRequired()])

```

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

```

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

```

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

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

```
1 def update_rocket_d1(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Update rocket_d1
7         rocket_d1_columns = cur.execute("PRAGMA table_info(rocket_details_1)").fetchall()
8         rocket_d1_columns_str = ",".join([column["name"] + "=? " for column
9         in rocket_d1_columns if column["name"] != "rocket_id"])
10
11         rocket_d1 = []
12         for column in rocket_d1_columns:
13             if column["name"] in request.form.keys():
14                 rocket_d1 += [request.form[column["name"]]]
15
16         rocket_id = rocket_d1[0]
17         rocket_d1 = rocket_d1[1:] + [rocket_id]
18
19         cur.execute(f'UPDATE rocket_details_1 SET {rocket_d1_columns_str}
20 WHERE rocket_id = ?', rocket_d1)
21         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:

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

SQL statement in database.py:

```
1 def get_cores():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             "SELECT * FROM cores"
7             ).fetchall()
```

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

To show datas in table

```
1 <header>
2     <button type="button" class="btn btn-warning btn" data-toggle="modal"
3     data-target="#add-core-modal" style="position: fixed; bottom: 5em; right
4     : 2.5em;">
5         New Core
6     </button>
7     <button type="button" class="btn btn-warning" data-toggle="modal" data-
8     target="#filter-core-modal" style="position: fixed; bottom: 2.5em; right
9     : 2.5em;">
10         Filter Core
11     </button>
12 <table class = "table table-hover table-sm">
13     <tr class = "table-active" style="text-align: center;">
14         <th>Serial</th>
15         <th>Status</th>
16         <th>Reuse Count</th>
17         <th>Block</th>
18         <th>Real Time Locating System Attempts</th>
19         <th>Real Time Locating System Landings</th>
20         <th>Autonomous Spaceport Drone Ship Attempts</th>
21         <th>Autonomous Spaceport Drone Ship Landings</th>
22         <th colspan="2">Edit</th>
23     </tr>
24     {% for core in cores %}
25     <tr class style="text-align: center;">
26         <td>
27             {{core["serial"]}}
28         </td>
29         <td>
30             {{core["status"]}}
31         </td>
32         <td>
33             {{core["reuse_count"]}}
34         </td>
35         <td>
36             {{core["block"]}}
37         </td>
38         <td>
39             {{core["rtls_attempts"]}}
40         </td>
41         <td>
42             {{core["rtls_landings"]}}
```

```

39         </td>
40         <td>
41             {{core["asds_attempts"]}}
42         </td>
43         <td>
44             {{core["asds_landings"]}}
45         </td>
46         <td>
47             <button type="button" class="btn btn-danger btn-sm" data-toggle
="modal" onclick="delete_core('{{ core.core_id }}')">
48                 Delete
49             </button>
50         </td>
51         <td>
52             <button type="button" class="btn btn-warning btn-sm" data-
toggle="modal" data-target="#update-core-modal-{{ core.core_id }}">
53                 Update
54             </button>
55         </td>
56     </tr>
57     {% endfor %}
58 </table>

```

2.2.8 Core Deleting

The view function to delete core in views.py:

```

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

```

Delete button for relevant line in table

```

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

```

SQL statement for deleting core in database.py:

```

1 def delete_core(core_id):
2     with sqlite3.connect(db_location) as con:
3         cursor = con.cursor()
4         cursor.execute("PRAGMA foreign_keys=ON")
5         query = "DELETE FROM cores WHERE (core_id = ?)"
6         cursor.execute(query, (core_id,))
7         con.commit()

```

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

```

1 function delete_core(core_id) {
2     document.location = '/delete_core?core_id=' + core_id
3 }

```

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:

```
1 class CoresForm(FlaskForm):
2     core_id = StringField()
3     serial = StringField()
4     status = StringField()
5     reuse_count = IntegerField()
6     block = IntegerField()
7     rtls_attempts = IntegerField()
8     rtls_landings = IntegerField()
9     asds_attempts = IntegerField()
10    asds_landings = IntegerField()
```

The view function to add core in views.py:

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

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

```
1 def add_core(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         core_columns = cur.execute("PRAGMA table_info(cores)").fetchall()
7
8         new_core = [str(current_app.config["core_id"])]
9         current_app.config["core_id"] += 1
10        for column in core_columns:
11            if column["name"] in request.form.keys():
12                new_core += [request.form[column["name"]]]
13
14        cur.execute(f'INSERT INTO cores VALUES ({",".join("'" * len(
15        core_columns))})', new_core)
16        con.commit()
```

Core adding form in cores.html:

```
1 <div class="modal-body"><form action="/add_core" method="POST" style="
2     display: grid;">
3     {{ core_form.csrf_token }}
4     <div class="field">
5         <label for="serial" class="form-label detail-field">Serial:</label>
6         <div class="control detail-field">
7             {{ core_form.serial(class="form-control-sm") }}
8         </div>
9     </div>
10    <div class="field">
11        <label for="status" class="form-label detail-field">Status:</label>
12        <div class="control detail-field">
13            {{ core_form.status(class="form-control-sm") }}
14        </div>
15    </div>
16    <div class="field">
17        <label for="reuse_count" class="form-label detail-field">Reuse Count:</
18        label>
```

```

17     <div class="control detail-field">
18         {{ core_form.reuse_count(class="form-control-sm") }}
19     </div>
20 </div>
21 <div class="field">
22     <label for="block" class="form-label detail-field">Block:</label>
23     <div class="control detail-field">
24         {{ core_form.block(class="form-control-sm") }}
25     </div>
26 </div>
27 <div class="field">
28     <label for="rtls_attempts" class="form-label detail-field">Real Time
29     Locating System Attempts:</label>
30     <div class="control detail-field">
31         {{ core_form.rtls_attempts(class="form-control-sm") }}
32     </div>
33 </div>
34 <div class="field">
35     <label for="rtls_landings" class="form-label detail-field">Real Time
36     Locating System Landings:</label>
37     <div class="control detail-field">
38         {{ core_form.rtls_landings(class="form-control-sm") }}
39     </div>
40 </div>
41 <div class="field">
42     <label for="asds_attempts" class="form-label detail-field">Autonomous
43     Spaceport Drone Ship Attempts:</label>
44     <div class="control detail-field">
45         {{ core_form.asds_attempts(class="form-control-sm") }}
46     </div>
47 </div>
48 <div class="field">
49     <label for="asds_landings" class="form-label detail-field">Autonomous
50     Spaceport Drone Ship Landings:</label>
51     <div class="control detail-field">
52         {{ core_form.asds_landings(class="form-control-sm") }}
53     </div>
54 </div>
55 <button type="submit" class="btn btn-warning">Add</button>
56 </form>
57 </div>

```

”New Core” button in the header

```

1     <button type="button" class="btn btn-warning btn" data-toggle="modal"
2     data-target="#add-core-modal" style="position: fixed; bottom: 5em; right
3     : 2.5em;">
4         New Core
5     </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:

```

1 class CoresForm(FlaskForm):
2     core_id = StringField()
3     serial = StringField()
4     status = StringField()
5     reuse_count = IntegerField()
6     block = IntegerField()

```

```

7     rtls_attempts = IntegerField()
8     rtls_landings = IntegerField()
9     asds_attempts = IntegerField()
10    asds_landings = IntegerField()

```

The view function to update core in views.py:

```

1 @views.route("/update_core", methods=["POST"])
2 @login_required
3 def update_core():
4     if current_user.is_admin:
5         database.update_core(request)
6     return redirect(url_for("views.cores"))

```

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

```

1 def update_core(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         core_columns = cur.execute("PRAGMA table_info(cores)").fetchall()
7         core_columns_str = ",".join([column["name"] + "=?" for column in
8             core_columns if column["name"] != "core_id"])
9
10        core = []
11        for column in core_columns:
12            if column["name"] in request.form.keys():
13                core += [request.form[column["name"]]]
14
15        core_id = core[0]
16        core = core[1:] + [core_id]
17
18        cur.execute(f'UPDATE cores SET {core_columns_str} WHERE core_id = ?',
19            core)
20        con.commit()

```

Core updating form in cores.html:

```

1 <div class="modal-body"><form action="/update_core" method="POST" style="
2     display: grid;">
3     {{ core_form.csrf_token }}
4     <div class="field" style="display: none">
5         <div class="control">
6             {{ core_form.core_id(value=core.core_id, class="form-control-sm") }}
7         </div>
8     </div>
9     <div class="field">
10        <label for="serial" class="form-label detail-field">Serial:</label>
11        <div class="control detail-field">
12            {{ core_form.serial(value=core.serial, class="form-control-sm") }}
13        </div>
14    </div>
15    <div class="field">
16        <label for="status" class="form-label detail-field">Status:</label>
17        <div class="control detail-field">
18            {{ core_form.status(value=core.status, class="form-control-sm") }}
19        </div>
20    </div>
21    <div class="field">
22        <label for="reuse_count" class="form-label detail-field">Reuse Count:</
23        label>
24        <div class="control detail-field">

```



```

23     {{ core_form.reuse_count(value=core.reuse_count,class="form-control-
    sm") }}
24     </div>
25 </div>
26 <div class="field">
27     <label for="block" class="form-label detail-field">Block:</label>
28     <div class="control detail-field">
29         {{ core_form.block(value=core.block,class="form-control-sm") }}
30     </div>
31 </div>
32 <div class="field">
33     <label for="rtls_attempts" class="form-label detail-field">Real Time
    Locating System Attempts:</label>
34     <div class="control detail-field">
35         {{ core_form.rtls_attempts(value=core.rtls_attempts,class="form-
    control-sm") }}
36     </div>
37 </div>
38 <div class="field">
39     <label for="rtls_landings" class="form-label detail-field">Real Time
    Locating System Landings:</label>
40     <div class="control detail-field">
41         {{ core_form.rtls_landings(value=core.rtls_landings,class="form-
    control-sm") }}
42     </div>
43 </div>
44 <div class="field">
45     <label for="asds_attempts" class="form-label detail-field">Autonomous
    Spaceport Drone Ship Attempts:</label>
46     <div class="control detail-field">
47         {{ core_form.asds_attempts(value=core.asds_attempts,class="form-
    control-sm") }}
48     </div>
49 </div>
50 <div class="field">
51     <label for="asds_landings" class="form-label detail-field">Autonomous
    Spaceport Drone Ship Landings:</label>
52     <div class="control detail-field">
53         {{ core_form.asds_landings(value=core.asds_landings,class="form-
    control-sm") }}
54     </div>
55 </div>
56 <button type="submit" class="btn btn-warning">Update</button>
57 </form>
58 </div>

```

Update button for relevant line in table

```

1     <td>
2         <button type="button" class="btn btn-warning btn-sm" data-
    toggle="modal" data-target="#update-core-modal-{{ core.core_id }}">
3             Update
4         </button>
5     </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:

```

1 class CoresForm(FlaskForm):

```

```

2     core_id = StringField()
3     serial = StringField()
4     status = StringField()
5     reuse_count = IntegerField()
6     block = IntegerField()
7     rtls_attempts = IntegerField()
8     rtls_landings = IntegerField()
9     asds_attempts = IntegerField()
10    asds_landings = IntegerField()

```

The view function to filter core in views.py:

```

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

```

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

```

1 def filter_core(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         print("filter_core executed")
6         query = "SELECT * FROM cores"
7         params = []
8         print(request.form.get('fserial'))
9         if request.form.get('fserial'):
10            if not params:
11                query += " WHERE"
12            user_serial = request.form['fserial']
13            query += " serial LIKE ?"
14            param_serial = "%" + user_serial + "%"
15            params.append(param_serial)
16            if request.form.get('fstatus'):
17                if not params:
18                    query += " WHERE"
19                else:
20                    query += " AND"
21                user_status = request.form['fstatus']
22                query += " status LIKE ?"
23                param_status = "%" + user_status + "%"
24                params.append(param_status)
25            if request.form.get('freuse_count') != "":
26                if not params:
27                    query += " WHERE"
28                else:
29                    query += " AND"
30                user_reuse_count = request.form['freuse_count']
31                query += " reuse_count LIKE ?"
32                param_reuse_count = "%" + user_reuse_count + "%"
33                params.append(param_reuse_count)
34
35            if request.form.get('fblock'):
36                if not params:
37                    query += " WHERE"
38                else:
39                    query += " AND"
40                user_block = request.form['fblock']
41                query += " block LIKE ?"
42                param_block = "%" + user_block + "%"

```

```

43     params.append(param_block)
44
45     if request.form.get('ftrls_attempts'):
46         if not params:
47             query += " WHERE"
48         else:
49             query += " AND"
50         user_rtls_attempts = request.form['ftrls_attempts']
51         query += " rtls_attempts LIKE ?"
52         param_rtls_attempts = "%" + user_rtls_attempts + "%"
53         params.append(param_rtls_attempts)
54
55     if request.form.get('ftrls_landings'):
56         if not params:
57             query += " WHERE"
58         else:
59             query += " AND"
60         user_rtls_landings = request.form['ftrls_landings']
61         query += " rtls_landings LIKE ?"
62         param_rtls_landings = "%" + user_rtls_landings + "%"
63         params.append(param_rtls_landings)
64
65     if request.form.get('fasds_attempts'):
66         if not params:
67             query += " WHERE"
68         else:
69             query += " AND"
70         user_asds_attempts = request.form['fasds_attempts']
71         query += " asds_attempts LIKE ?"
72         param_asds_attempts = "%" + user_asds_attempts + "%"
73         params.append(param_asds_attempts)
74     if request.form.get('fasds_landings'):
75         if not params:
76             query += " WHERE"
77         else:
78             query += " AND"
79         user_asds_landings = request.form['fasds_landings']
80         query += " asds_landings LIKE ?"
81         param_asds_landings = "%" + user_asds_landings + "%"
82         params.append(param_asds_landings)
83
84     print(query)
85     print(tuple(params))
86     filter = cur.execute(query, tuple(params)).fetchall()
87     return filter

```

Core filtering form in cores.html:

```

1 <div class="modal-body"><form action="/update_core" method="POST" style="
  display: grid;">
2   {{ core_form.csrf_token }}
3   <div class="field" style="display: none">
4     <div class="control">
5       {{ core_form.core_id(value=core.core_id, class="form-control-sm") }}
6     </div>
7   </div>
8   <div class="field">
9     <label for="serial" class="form-label detail-field">Serial:</label>
10    <div class="control detail-field">
11      {{ core_form.serial(value=core.serial, class="form-control-sm") }}
12    </div>
13  </div>

```

```

14 <div class="field">
15   <label for="status" class="form-label detail-field">Status:</label>
16   <div class="control detail-field">
17     {{ core_form.status(value=core.status,class="form-control-sm") }}
18   </div>
19 </div>
20 <div class="field">
21   <label for="reuse_count" class="form-label detail-field">Reuse Count:</label>
22   <div class="control detail-field">
23     {{ core_form.reuse_count(value=core.reuse_count,class="form-control-sm") }}
24   </div>
25 </div>
26 <div class="field">
27   <label for="block" class="form-label detail-field">Block:</label>
28   <div class="control detail-field">
29     {{ core_form.block(value=core.block,class="form-control-sm") }}
30   </div>
31 </div>
32 <div class="field">
33   <label for="rtls_attempts" class="form-label detail-field">Real Time
34   Locating System Attempts:</label>
35   <div class="control detail-field">
36     {{ core_form.rtls_attempts(value=core.rtls_attempts,class="form-control-sm") }}
37   </div>
38 </div>
39 <div class="field">
40   <label for="rtls_landings" class="form-label detail-field">Real Time
41   Locating System Landings:</label>
42   <div class="control detail-field">
43     {{ core_form.rtls_landings(value=core.rtls_landings,class="form-control-sm") }}
44   </div>
45 </div>
46 <div class="field">
47   <label for="asds_attempts" class="form-label detail-field">Autonomous
48   Spaceport Drone Ship Attempts:</label>
49   <div class="control detail-field">
50     {{ core_form.asds_attempts(value=core.asds_attempts,class="form-control-sm") }}
51   </div>
52 </div>
53 <div class="field">
54   <label for="asds_landings" class="form-label detail-field">Autonomous
55   Spaceport Drone Ship Landings:</label>
56   <div class="control detail-field">
57     {{ core_form.asds_landings(value=core.asds_landings,class="form-control-sm") }}
58   </div>
59 </div>
60 <button type="submit" class="btn btn-warning">Update</button>
61 </form>
62 </div>

```

Filter button in the header

```

1 <button type="button" class="btn btn-warning" data-toggle="modal" data-
2 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`

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

`login.html`

```
1     <form action="/login" method="POST">
2         {{ form.csrf_token }}
3         <div class="field" style="margin: 90px 0px 0px 80px;">
4             <label for="username" class="form-label" style="color:white;">Username:</label>
5             <div class="control">
6                 {{ form.username(required=True, class="form-control-lg") }}
7             </div>
8         </div>
9         <br>
10        <div class="field" style="margin: 0px 0px 0px 80px;">
11            <label for="password" class="form-label" style="color:white;">Password:</label>
12            <div class="control">
13                {{ form.password(required=True, class="form-control-lg") }}
14            </div>
15        </div>
16
17        <button type="submit" class="btn btn-warning" style="margin: 20px 20px 90px;">Log in</button>
18    </form>
```

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

`views.py`

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

```

5     user = users.get_user(username)
6
7     if user is not None:
8         password = form.data["password"]
9         if passlib.hash.pbkdf2_sha256.verify(password, user.password):
10             login_user(user)
11             flash("Welcome to SpaceXhibit!")
12
13             return redirect(url_for("views.home"))
14         else:
15             flash("Wrong password.")
16     else:
17         flash("You're not on the guest list. Why don't you sign up?")
18
19     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

```

1 def get_user(username):
2     user_db = sqlite3.connect(user_db_location)
3     user_db.row_factory = sqlite3.Row
4     user_data = user_db.cursor().execute("SELECT * FROM users WHERE
5     username = ?", [username]).fetchall()
6
7     if len(user_data) == 1:
8         return User(user_data[0]["username"], user_data[0]["password"],
9         user_data[0]["type"])
10    else:
11        return None

```

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

forms.py

```

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

```

signup.html

```

1 <form action="/signup" method="POST">
2     {{ form.csrf_token }}
3     <div class="field" style="margin: 90px 0px 0px 80px;">
4         <label for="username" class="form-label label-color" style="
5         color:white;">Username:</label>
6         <div class="control">
7             {{ form.username(required=True, class="form-control-lg") }}
8         </div>
9     <br>
10    <div class="field" style="margin: 0px 0px 0px 80px;">
11        <label for="password" class="form-label" style="color:white;">
12        Password:</label>
13        <div class="control">
14            {{ form.password(required=True, class="form-control-lg") }}
15        </div>

```

```

15     </div>
16     <br>
17     <div class="field" style="margin: 0px 0px 0px 80px;">
18         <label for="superfan" class="form-label" style="color:white;">
Are you a superfan of SpaceX?</label>
19         <div class="control" style="color:white;">
20             {{ form.superfan(required=True, class="form-control-lg list
-style-none") }}
21         </div>
22     </div>
23     <div class="field" style="margin: 0px 0px 0px 80px;">
24         <div class="control" style="color:white;">
25             <input type="checkbox" name="wants_admin">
26             <label for="wants_admin" class="form-label" style="color:
white;">I would like to be an admin.</label>
27         </div>
28     </div>
29
30     <button type=submit class="btn btn-warning" style="margin: 10px 20
px 20px 80px; color:white;">Sign up</button>
31 </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

```

1 def add_user(username, password, type):
2     user_db = sqlite3.connect(user_db_location)
3     user_db.row_factory = sqlite3.Row
4
5     user_db.cursor().execute(
6         'INSERT INTO users (username, password, type) VALUES (?, ?, ?)',
7         (username, password, type))
8     user_db.commit()

```

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

views.py

```

1 def signup():
2     form = forms.SignupForm()
3     if form.validate_on_submit():
4         username = form.data["username"]
5         user = users.get_user(username)
6
7         if user is None:
8             password = form.data["password"]
9             hashed_password = \
10                 passlib.hash.pbkdf2_sha256.hash(password)
11
12             user_type = "regular"
13             deserves_adminhood = True if form.data["superfan"] == "True"
14         else False
15             if deserves_adminhood and "wants_admin" in request.form.keys():
16                 user_type = "admin"
17
18             users.add_user(username, hashed_password, user_type)
19
20             flash("You're all set, you can log in.")
21             return redirect(url_for("views.login"))
22     else:
23         flash("That username is taken, please choose another.")

```

```

23
24     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

```

1 @views.route('/rockets', methods=['GET', 'POST'])
2 def rockets():
3     rocket_data = database.get_rockets()
4     rocket_d1_data = database.get_rocket_d1()
5     rocket_d2_data = database.get_rocket_d2()
6     rocket_image_data = database.get_rocket_image()
7
8     inexistent_rocket_d1 = []
9     inexistent_rocket_d2 = []
10    inexistent_rocket_image = []
11    inexistent_rocket_d1_dict = []
12    inexistent_rocket_d2_dict = []
13    inexistent_rocket_image_dict = []
14
15    present = False
16    for rocket in rocket_data:
17        for rocket_d1 in rocket_d1_data:
18            if rocket["rocket_id"] == rocket_d1["rocket_id"]:
19                present = True
20                break
21        if not present:
22            inexistent_rocket_d1 += [rocket["rocket_id"]]
23            inexistent_rocket_d1_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]]
24            present = False
25
26        for rocket_d2 in rocket_d2_data:
27            if rocket["rocket_id"] == rocket_d2["rocket_id"]:
28                present = True
29                break
30        if not present:
31            inexistent_rocket_d2 += [rocket["rocket_id"]]
32            inexistent_rocket_d2_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]]
33            present = False
34
35        for rocket_image in rocket_image_data:
36            if rocket["rocket_id"] == rocket_image["rocket_id"]:
37                present = True
38                break
39        if not present:
40            inexistent_rocket_image += [rocket["rocket_id"]]
41            inexistent_rocket_image_dict += [{"rocket_id": rocket["rocket_id"], "name": rocket["name"]}]]
42            present = False
43
44    return render_template("rockets.html", rockets=rocket_data,
45                           rocket_d1s=rocket_d1_data, rocket_d2s=rocket_d2_data, rocket_images=rocket_image_data,
46                           inexistent_d1=inexistent_rocket_d1, inexistent_d2=inexistent_rocket_d2, inexistent_image=inexistent_rocket_image,

```



```

47     inexistent_d1_dict=inexistent_rocket_d1_dict, inexistent_d2_dict=
inexistent_rocket_d2_dict, inexistent_image_dict=
inexistent_rocket_image_dict,
48     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

```

1 def get_rockets():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             "SELECT * FROM rockets"
7         ).fetchall()

```

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

rockets.html

```

1 <table class = "table table-hover table-sm">
2     <thead>
3         <tr class = "table-active">
4             <th></th>
5             <th>Name</th>
6             <th>Type</th>
7             <th>Status</th>
8             <th>Country of Origin</th>
9             <th>Company</th>
10            <th colspan="2">Details</th>
11            <th colspan="2">Editing</th>
12        </tr>
13    </thead>
14    <tbody>
15        {% for rocket in rockets %}
16        <tr class>
17            <td>
18                {% if rocket["rocket_id"] in inexistent_image %}
19                <button type="button" class="btn btn-sm" data-toggle="modal
" data-target="#add-rocket-image-modal-{{ rocket.rocket_id }}">
20
21                </button>
22                {% else %}
23                <button type="button" class="btn btn-sm" data-toggle="modal
" data-target="#rocket-image-modal-{{ rocket.rocket_id }}">
24
25                </button>
26                {% endif %}
27            </td>
28            <td> {{rocket["name"]}} </td>
29            <td> {{rocket["type"]}} </td>
30            <td> {{ "Active" if rocket["active"] == "True" else "
Decommissioned" }} </td>
31            <td> {{rocket["country"]}} </td>
32            <td> {{rocket["company"]}} </td>
33
34            <td>
35                {% if rocket["rocket_id"] in inexistent_d1 %}

```

```

36         <button type="button" class="btn btn-warning btn-sm" data-
toggle="modal" data-target="#add-rocket-details-1-modal-{{ rocket.
rocket_id }}">
37             Add Engine Info
38         </button>
39         {% else %}
40         <button type="button" class="btn btn-warning btn-sm" data-
toggle="modal" data-target="#rocket-details-1-modal-{{ rocket.rocket_id
}}">
41             Engine Info
42         </button>
43         {% endif %}
44     </td>
45     <td>
46         {% if rocket["rocket_id"] in inexistent_d2 %}
47         <button type="button" class="btn btn-warning btn-sm" data-
toggle="modal" data-target="#add-rocket-details-2-modal-{{ rocket.
rocket_id }}">
48             Add Measurements
49         </button>
50         {% else %}
51         <button type="button" class="btn btn-warning btn-sm" data-
toggle="modal" data-target="#rocket-details-2-modal-{{ rocket.rocket_id
}}">
52             Measurements
53         </button>
54         {% endif %}
55     </td>
56     <td>
57         <button type="button" class="btn btn-warning btn-sm" data-
toggle="modal" data-target="#update-rocket-modal-{{ rocket.rocket_id }}"
>
58             Update
59         </button>
60     </td>
61     <td>
62         {% if current_user.is_admin %}
63         <button type="button" class="btn btn-danger btn-sm" data-
toggle="modal"
64             onclick="delete_rocket('{{ rocket.rocket_id }}')">
65             Delete
66         </button>
67         {% endif %}
68     </td>
69
70
71 </tr>
72 {% endfor %}
73 </tbody>
74 </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

```

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

```

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

rockets.html

```
1
2 <div class="modal fade" id="add-rocket-modal" tabindex="-1" role="dialog"
  aria-labelledby="add-rocket-modal-label" aria-hidden="true">
3   <div class="modal-dialog" role="document">
4     <div class="modal-content">
5       <div class="modal-header">
6         <h5 class="modal-title" id="rocket-modal-label"> Add Rocket </h5>
7       </div>
8       <div class="modal-body"><form action="/add_rocket" method="POST"
  style="display: grid;">
9         {{ formM.csrf_token }}
10        <div class="field">
11          <label for="name" class="form-label detail-field">Name:</label>
12          <div class="control detail-field">
13            {{ formM.name(class="form-control-sm") }}
14          </div>
15        </div>
16        <div class="field">
17          <label for="type" class="form-label detail-field">Type:</label>
18          <div class="control detail-field">
19            {{ formM.type(class="form-control-sm") }}
20          </div>
21        </div>
22        <div class="field">
23          <label for="active" class="form-label detail-field">Active:</
  label>
24          <div class="control detail-field">
25            {{ formM.active(class="form-control-sm") }}
26          </div>
27        </div>
28        <div class="field">
29          <label for="country" class="form-label detail-field">Country:</
  label>
30          <div class="control detail-field">
31            {{ formM.country(class="form-control-sm") }}
32          </div>
33        </div>
34        <div class="field">
35          <label for="company" class="form-label detail-field">Company:</
  label>
36          <div class="control detail-field">
37            {{ formM.company(class="form-control-sm") }}
38          </div>
39        </div>
40
41        <button type="submit" class="btn btn-warning">Add</button>
42      </form>
43    </div>
44    <div class="modal-footer">
45      <button type="button" class="btn btn-secondary" data-dismiss="modal
  ">Close</button>
46    </div>
47  </div>
48 </div>
49 </div>
```

forms.py

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

```

2     rocket_id = StringField()
3     name = StringField()
4     type = StringField()
5     active = SelectField(choices=[("True", "Active"), ("False", "
Decommissioned")])
6     country = StringField()
7     company = StringField()

```

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

`views.py`

```

1 def add_rocket():
2     form = forms.RocketForm()
3     if form.validate_on_submit():
4         database.add_rocket(request)
5     return redirect(url_for("views.rockets"))

```

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

`database.py`

```

1 def add_rocket(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new rocket
7         rocket_columns = cur.execute("PRAGMA table_info(rockets)").fetchall
8         ()
9
10        new_rocket = [str(current_app.config["rocket_id"])] # Get next ID
11        current_app.config["rocket_id"] += 1
12        for column in rocket_columns:
13            if column["name"] in request.form.keys():
14                new_rocket += [request.form[column["name"]]]
15
16        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`

```

1
2 {% for rocket in rockets %}
3
4 <div class="modal fade" id="update-rocket-modal-{{ rocket.rocket_id }}"
5     tabindex="-1" role="dialog" aria-labelledby="update-rocket-modal-label
6     -{{ rocket.rocket_id }}" aria-hidden="true">
7     <div class="modal-dialog" role="document">
8         <div class="modal-content">
9             <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") }}

```

```

54         </div>
55     </div>
56
57     <button type=submit class="btn btn-warning">Update</button>
58 </form>
59 </div>
60 <div class="modal-footer">
61     <button type="button" class="btn btn-secondary" data-dismiss="
modal">Close</button>
62 </div>
63 </div>
64 </div>
65 </div>
66
67 {% endfor %}

```

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

```

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

```

The database operations are performed as follows:
database.py

```

1     con.row_factory = sqlite3.Row
2     cur = con.cursor()
3
4     # Update rocket
5     rocket_columns = cur.execute("PRAGMA table_info(rockets)").fetchall
6     ()
7     rocket_columns_str = ",".join([column["name"] + "=? " for column in
8     rocket_columns if column["name"] != "rocket_id"])
9
10    rocket = []
11    for column in rocket_columns:
12        if column["name"] in request.form.keys():
13            rocket += [request.form[column["name"]]]
14
15    rocket_id = rocket[0]
16    rocket = rocket[1:] + [rocket_id]
17
18    cur.execute(f'UPDATE rockets SET {rocket_columns_str} WHERE
19    rocket_id = ?', rocket)
20    con.commit()
21
22 def update_rocket_d1(request):
23     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

```

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

```

```

2 @login_required
3 def delete_rocket():
4     if current_user.is_admin:
5         database.delete_rocket(request.args.get("rocket_id"))
6     else:
7         flash("Please do not poke around the exhibit.")
8     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

```

1     cursor = con.cursor()
2     cursor.execute("PRAGMA foreign_keys=ON") # This allows for
3     cascading to details
4     query = "DELETE FROM rockets WHERE (rocket_id = ?)"
5     cursor.execute(query, (rocket_id,))
6     con.commit()
7 def delete_rocket_d1(rocket_id):
8     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

```

1
2 {% for rocket_d2 in rocket_d2s %}
3 <div class="modal fade" id="update-rocket-details-2-modal-{{ rocket_d2.
4     rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="update-rocket
5     -details-2-modal-label-{{ rocket_d2.rocket_id }}" aria-hidden="true">
6     <div class="modal-dialog" role="document">
7         <div class="modal-content">
8             <div class="modal-header">
9                 <h5 class="modal-title" id="update-rocket-details-2-modal-label
10                 -{{ rocket_d2.rocket_id }}"> Update {{rocket_d2.name}} Measurements </h5
11             >
12             </div>
13             <div class="modal-body">
14                 <form action="/update_rocket_detail_2" method="POST" style="
15                 display: grid;">
16                     {{ formD2.csrf_token }}
17                     <div class="field" style="display: none">
18                         <div class="control">
19                             {{ formD2.rocket_id(value=rocket_d2.rocket_id, class="
20                             form-control-sm") }}
21                         </div>
22                     </div>
23                     <div class="field">
24                         <label for="height_mt" class="form-label detail-field">Height
25                         (m):</label>
26                         <div class="control detail-field">
27                             {{ formD2.height_mt(value=rocket_d2.height_mt, class="
28                             form-control-sm") }}
29                         </div>

```

```

22         </div>
23         <div class="field">
24             <label for="height_ft" class="form-label detail-field">Height
25             (ft):</label>
26             <div class="control detail-field">
27                 {{ formD2.height_ft(value=rocket_d2.height_ft, class="
28 form-control-sm")}}
29             </div>
30         </div>
31         <div class="field">
32             <label for="diameter_mt" class="form-label detail-field">
33 Diameter (m):</label>
34             <div class="control detail-field">
35                 {{ formD2.diameter_mt(value=rocket_d2.diameter_mt, class=
36 "form-control-sm")}}
37             </div>
38         </div>
39         <div class="field">
40             <label for="diameter_ft" class="form-label detail-field">
41 Diameter (ft):</label>
42             <div class="control detail-field">
43                 {{ formD2.diameter_ft(value=rocket_d2.diameter_ft, class=
44 "form-control-sm")}}
45             </div>
46         </div>
47         <div class="field">
48             <label for="mass_kg" class="form-label detail-field">Mass (kg
49 ):</label>
50             <div class="control detail-field">
51                 {{ formD2.mass_kg(value=rocket_d2.mass_kg, class="form-
52 control-sm")}}
53             </div>
54         </div>
55         <div class="field">
56             <label for="mass_lb" class="form-label detail-field">Mass (lb
57 ):</label>
58             <div class="control detail-field">
59                 {{ formD2.mass_lb(value=rocket_d2.mass_lb, class="form-
60 control-sm")}}
61             </div>
62         </div>
63     </div>
64     <button type="submit" class="btn btn-warning">Update</button>
65 </form>
66 </div>
67 <div class="modal-footer">
68     <button type="button" class="btn btn-secondary" data-dismiss="
69 modal">Close</button>
70 </div>
71 </div>
72 </div>
73 {% 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

```

1     con.row_factory = sqlite3.Row
2     cur = con.cursor()
3     return cur.execute(

```



```

4         """SELECT * FROM rocket_details_2
5         JOIN (SELECT rocket_id, name FROM rockets) AS rocket_names
6         ON rocket_details_2.rocket_id = rocket_names.rocket_id"""
7         ).fetchall()
8 def get_rocket_image():
9     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

```

1
2 {% for rocket in inexistent_d2_dict %}
3 <div class="modal fade" id="add-rocket-details-2-modal-{{ rocket.rocket_id
4 }}" tabindex="-1" role="dialog" aria-labelledby="add-rocket-details-2-
5 modal-label-{{ rocket.rocket_id }}" aria-hidden="true">
6     <div class="modal-dialog" role="document">
7         <div class="modal-content">
8             <div class="modal-header">
9                 <h5 class="modal-title" id="rocket-details-2-modal-label-{{
10 rocket.rocket_id }}"> Add {{rocket.name}} Measurements </h5>
11             </div>
12             <div class="modal-body">
13                 <form action="/add_rocket_detail_2" method="POST" style="display:
14 grid;">
15                     {{ formD2.csrf_token }}
16                     <div class="field" style="display: none">
17                         <div class="control">
18                             {{ formD2.rocket_id(value=rocket.rocket_id, class="form-
19 control-sm") }}
20                         </div>
21                     </div>
22                     <div class="field">
23                         <label for="height_mt" class="form-label detail-field">Height
24 (m):</label>
25                         <div class="control detail-field">
26                             {{ formD2.height_mt(class="form-control-sm") }}
27                         </div>
28                     </div>
29                     <div class="field">
30                         <label for="height_ft" class="form-label detail-field">Height
31 (ft):</label>
32                         <div class="control detail-field">
33                             {{ formD2.height_ft(class="form-control-sm") }}
34                         </div>
35                     </div>
36                     <div class="field">
37                         <label for="diameter_mt" class="form-label detail-field">
38 Diameter (m):</label>
39                         <div class="control detail-field">
40                             {{ formD2.diameter_mt(class="form-control-sm") }}
41                         </div>
42                     </div>
43                     <div class="field">
44                         <label for="diameter_ft" class="form-label detail-field">
45 Diameter (ft):</label>

```

```

37         <div class="control detail-field">
38             {{ formD2.diameter_ft(class="form-control-sm") }}
39         </div>
40     </div>
41     <div class="field">
42         <label for="mass_kg" class="form-label detail-field">Mass (kg
43     ):</label>
44         <div class="control detail-field">
45             {{ formD2.mass_kg(class="form-control-sm") }}
46         </div>
47     <div class="field">
48         <label for="mass_lb" class="form-label detail-field">Mass (lb
49     ):</label>
50         <div class="control detail-field">
51             {{ formD2.mass_lb(class="form-control-sm") }}
52         </div>
53     </div>
54     <button type="submit" class="btn btn-warning">Add</button>
55 </form>
56 </div>
57 <div class="modal-footer">
58     <button type="button" class="btn btn-secondary" data-dismiss="
59 modal">Close</button>
60 </div>
61 </div>
62 </div>
63 {% endfor %}

```

This modal's form is defined as follows:
forms.py

```

1 class RocketD2Form(FlaskForm):
2     rocket_id = StringField()
3     height_mt = DecimalField(validators=[validators.InputRequired()])
4     height_ft = DecimalField(validators=[validators.InputRequired()])
5     diameter_mt = DecimalField(validators=[validators.InputRequired()])
6     diameter_ft = DecimalField(validators=[validators.InputRequired()])
7     mass_kg = IntegerField(validators=[validators.InputRequired()])
8     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

```

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

```

The database connection is then handled as follows:
database.py

```

1     con.row_factory = sqlite3.Row
2     cur = con.cursor()
3
4     # Add new rocket_d2
5     rocket_d2_columns = cur.execute("PRAGMA table_info(rocket_details_2
6 )").fetchall()

```

```

6
7     new_rocket_d2 = []
8     for column in rocket_d2_columns:
9         if column["name"] in request.form.keys():
10             new_rocket_d2 += [request.form[column["name"]]]
11
12     cur.execute(f'INSERT INTO rocket_details_2 VALUES ({",".join("'" *
13         len(rocket_d2_columns)})', new_rocket_d2)
14     con.commit()
15 def add_rocket_image(request):
16     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

```

1
2 {% for rocket_d2 in rocket_d2s %}
3 <div class="modal fade" id="update-rocket-details-2-modal-{{ rocket_d2.
4     rocket_id }}" tabindex="-1" role="dialog" aria-labelledby="update-rocket
5     -details-2-modal-label-{{ rocket_d2.rocket_id }}" aria-hidden="true">
6     <div class="modal-dialog" role="document">
7         <div class="modal-content">
8             <div class="modal-header">
9                 <h5 class="modal-title" id="update-rocket-details-2-modal-label
10                 -{{ rocket_d2.rocket_id }}"> Update {{rocket_d2.name}} Measurements </h5
11             >
12             </div>
13             <div class="modal-body">
14                 <form action="/update_rocket_detail_2" method="POST" style="
15                 display: grid;">
16                     {{ formD2.csrf_token }}
17                     <div class="field" style="display: none">
18                         <div class="control">
19                             {{ formD2.rocket_id(value=rocket_d2.rocket_id, class="
20                 form-control-sm") }}
21                         </div>
22                     </div>
23                     <div class="field">
24                         <label for="height_mt" class="form-label detail-field">Height
25                         (m):</label>
26                         <div class="control detail-field">
27                             {{ formD2.height_mt(value=rocket_d2.height_mt, class="
28                 form-control-sm") }}
29                         </div>
30                     </div>
31                     <div class="field">
32                         <label for="height_ft" class="form-label detail-field">Height
33                         (ft):</label>
34                         <div class="control detail-field">
35                             {{ formD2.height_ft(value=rocket_d2.height_ft, class="
36                 form-control-sm") }}
37                         </div>
38                     </div>
39                 </form>
40             </div>
41         </div>
42     </div>
43 </div>

```

```

29         <div class="field">
30             <label for="diameter_mt" class="form-label detail-field">
Diameter (m):</label>
31             <div class="control detail-field">
32                 {{ formD2.diameter_mt(value=rocket_d2.diameter_mt, class=
"form-control-sm") }}
33             </div>
34         </div>
35         <div class="field">
36             <label for="diameter_ft" class="form-label detail-field">
Diameter (ft):</label>
37             <div class="control detail-field">
38                 {{ formD2.diameter_ft(value=rocket_d2.diameter_ft, class=
"form-control-sm") }}
39             </div>
40         </div>
41         <div class="field">
42             <label for="mass_kg" class="form-label detail-field">Mass (kg
):</label>
43             <div class="control detail-field">
44                 {{ formD2.mass_kg(value=rocket_d2.mass_kg, class="form-
control-sm") }}
45             </div>
46         </div>
47         <div class="field">
48             <label for="mass_lb" class="form-label detail-field">Mass (lb
):</label>
49             <div class="control detail-field">
50                 {{ formD2.mass_lb(value=rocket_d2.mass_lb, class="form-
control-sm") }}
51             </div>
52         </div>
53
54         <button type="submit" class="btn btn-warning">Update</button>
55     </form>
56 </div>
57 <div class="modal-footer">
58     <button type="button" class="btn btn-secondary" data-dismiss="
modal">Close</button>
59 </div>
60 </div>
61 </div>
62 </div>
63 {% endfor %}

```

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

views.py

```

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

```

The database operation is as follows:

database.py

```

1 con.row_factory = sqlite3.Row
2 cur = con.cursor()
3

```

```

4      # Update rocket_d2
5      rocket_d2_columns = cur.execute("PRAGMA table_info(rocket_details_2
6  )").fetchall()
7      rocket_d2_columns_str = ",".join([column["name"] + "=?" for column
8  in rocket_d2_columns if column["name"] != "rocket_id"])
9
10     rocket_d2 = []
11     for column in rocket_d2_columns:
12         if column["name"] in request.form.keys():
13             rocket_d2 += [request.form[column["name"]]]
14
15     rocket_id = rocket_d2[0]
16     rocket_d2 = rocket_d2[1:] + [rocket_id]
17
18     cur.execute(f'UPDATE rocket_details_2 SET {rocket_d2_columns_str}
19 WHERE rocket_id = ?', rocket_d2)
20     con.commit()
21
22 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

```

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

```

This deletion is then performed on the database as follows:

database.py

```

1     cursor = con.cursor()
2     query = "DELETE FROM rocket_details_2 WHERE (rocket_id = ?)"
3     cursor.execute(query, (rocket_id,))
4     con.commit()
5 def delete_rocket_image(rocket_id):
6     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 accessed through their own modals which are generated as follows:

rockets.html

```

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

```

```

7         <h5 class="modal-title" id="rocket-image-modal-label-{{
rocket_image.rocket_id }}"> {{ rocket_image.name }} Photo </h5>
8     </div>
9     <div class="modal-body">
10         
11     </div>
12     <div class="modal-footer">
13         <button type="button" onclick="delete_rocket_image('{{
rocket_image.rocket_id }}')"
14             class="btn btn-danger" data-dismiss="modal">Delete Photo</button
>
15         <button type="button" class="btn btn-secondary" data-dismiss="
modal">Close</button>
16     </div>
17 </div>
18 </div>
19 </div>
20 {% 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

```

1     con.row_factory = sqlite3.Row
2     cur = con.cursor()
3     data = cur.execute(
4         """SELECT * FROM rocket_images
5         JOIN (SELECT rocket_id, name FROM rockets) AS rocket_names
6         ON rocket_images.rocket_id = rocket_names.rocket_id"""
7     ).fetchall()
8     for row in data:
9         try:
10             with open("/static/rocket_images/" + row["rocket_id"] + ".
png", "wb+") as image_file:
11                 image_file.write(row["rocket_image"])
12             except FileNotFoundError:
13                 with open("website/static/rocket_images/" + row["rocket_id"
] + ".png", "wb+") as image_file:
14                     image_file.write(row["rocket_image"])
15             return data
16
17 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

```

1
2 {% for rocket in inexistent_image_dict %}
3 <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">
4     <div class="modal-dialog" role="document">
5         <div class="modal-content">
6             <div class="modal-header">
7                 <h5 class="modal-title" id="rocket-image-modal-label-{{ rocket.
rocket_id }}"> Add {{rocket.name}} Photo </h5>
8             </div>

```

```

9         <div class="modal-body">
10             <form action="/add_rocket_image" method="POST" enctype="multipart
11 /form-data" style="display: grid;">
12                 {{ formI.csrf_token }}
13                 <div class="field" style="display: none">
14                     <div class="control">
15                         {{ formI.rocket_id(value=rocket.rocket_id, class="form-
16 control-sm") }}
17                     </div>
18                 </div>
19                 <div class="field">
20                     <div class="control detail-field">
21                         {{ formI.rocket_image(class="form-control") }}
22                     </div>
23                 </div>
24                 <button type=submit class="btn btn-warning">Add</button>
25             </form>
26             <div class="modal-footer">
27                 <button type="button" class="btn btn-secondary" data-dismiss="
28 modal">Close</button>
29             </div>
30         </div>
31 </div>
32 {% endfor %}

```

The form for this modal is as so:
forms.py

```

1 class RocketImageForm(FlaskForm):
2     rocket_id = StringField()
3     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

```

1 @views.route('/add_rocket_image', methods=['POST'])
2 def add_rocket_image():
3     form = forms.RocketImageForm()
4     if form.validate_on_submit():
5         database.add_rocket_image(request)
6     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

```

1     con.row_factory = sqlite3.Row
2     cur = con.cursor()
3
4     # Add new rocket_image
5
6     new_rocket_image = [request.form["rocket_id"]]
7
8     data = request.files["rocket_image"].read()
9     print(data)
10
11     new_rocket_image += [data]
12

```

```

13         cur.execute(f'INSERT INTO rocket_images VALUES (?, ?)',
14             new_rocket_image)
15         con.commit()
16 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

```

1 @views.route('/delete_rocket_image', methods=['GET'])
2 def delete_rocket_image():
3     database.delete_rocket_image(request.args.get("rocket_id"))
4     return redirect(url_for("views.rocket"))

```

The specified rocket is deleted as so:

database.py

```

1     cursor = con.cursor()
2     query = "DELETE FROM rocket_images WHERE (rocket_id = ?)"
3     cursor.execute(query, (rocket_id,))
4     con.commit()
5
6 # 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

```

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

```

Launchpad information is pulled from the database as follows:

database.py

```

1     con.row_factory = sqlite3.Row
2     cur = con.cursor()
3     return cur.execute(
4         "SELECT * FROM launchpads ORDER BY name"
5     )
6
7 # Payloads

```

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

launchpads.html

```

1 <table class = "table table-hover table-sm table-bordered">
2     <thead>
3         <tr class = "table-active">
4             <th>Name</th>

```



```

5         <th>Full Name</th>
6         <th>Status</th>
7         <th>Locality</th>
8         <th>Region</th>
9         <th>Timezone</th>
10        <th>Latitude</th>
11        <th>Longitude</th>
12    </tr>
13 </thead>
14 <tbody>
15     {% for launchpad in launchpads %}
16     <tr class>
17         <td> {{ launchpad["name"] }} </td>
18         <td> {{ launchpad["full_name"] }} </td>
19         <td> {{ launchpad["status"] }} </td>
20         <td> {{ launchpad["locality"] }} </td>
21         <td> {{ launchpad["region"] }} </td>
22         <td> {{ launchpad["timezone"] }} </td>
23         <td> {{ launchpad["latitude"] }} </td>
24         <td> {{ launchpad["longitude"] }} </td>
25     </tr>
26     {% endfor %}
27 </tbody>
28 </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.

```
1 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/
bootstrap.min.css" rel="stylesheet"
2 integrity="sha384-
rbsA2VBKQhggwzxH7pPCaAq046MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65"
3 crossorigin="anonymous">
4 <link
5 rel="stylesheet"
6 href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css"
7 crossorigin="anonymous"
8 />
9 <!-- JavaScript Bundle with Popper -->
10 <script defer src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js
/bootstrap.bundle.min.js"
11 integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hpcxyD9F7jL+jjXkk+
Q2h455rYXK/7HAuoJl+0I4"
12 crossorigin="anonymous"></script>
13 <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.

```
1 <nav class="navbar navbar-expand-lg navbar-light bg-warning" style="
padding-left: 10em">
2 <button
3 class="navbar-toggler"
4 type="button"
5 data-toggle="collapse"
6 data-target="#navbar"
7 >
8 <span class="navbar-toggler-icon"></span>
9 </button>
10 <svg xmlns="http://www.w3.org/2000/svg" width="25" height="25" fill="
currentColor" class="bi bi-rocket" viewBox="0 0 16 16">
11 <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"/>
12 <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 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 .931l1.125-1.5c
.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-.635.067.03.132.056.196.083.213.09.413.174.627.282.332.17.586.348.752.57
11.125 1.5a.5.5 0 0 0 .9-.3v-3.298a2.5 2.5 0 0 0-.548-1.562l-1.499-1.83
ZM12 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"/>
13 <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"/>
14 </svg>
15 <div class="collapse navbar-collapse" id="navbar">
16 <div class="navbar-nav">
17 <a class="nav-item nav-link" id="home" href="/home">Home</a>
18 <a class="nav-item nav-link" id="launches" href="/launches">
Launches</a>
19 <a class="nav-item nav-link" id="launchpads" href="/launchpads">
Launchpads</a>
20 <a class="nav-item nav-link" id="rockets" href="/rockets">Rockets
</a>
21 <a class="nav-item nav-link" id="ships" href="/ships">Ships</a>
22 <a class="nav-item nav-link" id="payloads" href="/payloads">
Payloads</a>
23 <a class="nav-item nav-link" id="cores" href="/cores">Cores</a>
24 <a class="nav-item nav-link" id="capsules" href="/capsules">
Capsules</a>
25
26 {% if not current_user.is_authenticated %}
27 <a class="nav-item nav-link pad-left" href="{% url_for('views.
login') %}">Login</a>
28 {% else %}
29 <p class="nav-item nav-link pad-left" style="margin: 0;">You
are logged in as {% current_user.username %}.</p>
30 <a class="nav-item nav-link pad-left" href="{% url_for('views.
logout') %}">Logout</a>
31 {% endif %}
32
33 </div>
34 </div>
35 </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.

```

1
2 <div class = "container">
3 <div class = "row">
4 <div class ="col">
5 <div class="card" style="width: 20rem;">
6 <div class="card-body">
7 <h5 class="card-title">Search</h5>
8 <p class="card-text">You can make extensive searches
about space missions conducted by SpaceX with our website!
9 You can get information on: </p>
10 </div>
11 <ul class="list-group list-group-flush">
12 <li class="list-group-item"><ul><li>Launches</li></ul><
/li>
13 <li class="list-group-item"><ul><li>Launchpads</li></ul>
></li>
14 <li class="list-group-item"><ul><li>Rockets</li></ul></
li>

```

```

15         <li class="list-group-item"><ul><li>Ships</li></ul></li>
16     >
17         <li class="list-group-item"><ul><li>Payloads</li></ul><
18 /li>
19         <li class="list-group-item"><ul><li>Cores</li></ul></li>
20 >
21         <li class="list-group-item"><ul><li>Capsules</li></ul><
22 /li>
23     </ul>
24 </div>
25 </div>
26 <div class = "col">
27     <div class="card" style="width: 20rem;">
28         <div class="card-body">
29             <h5 class="card-title">Edit</h5>
30             <p class="card-text">You can change our database with the
31 simple tools we provide you!
32             Here is a list of what you can do: </p>
33         </div>
34         <ul class="list-group list-group-flush">
35             <li class="list-group-item"><ul><li>Create entries</li></ul>
36 ></li>
37             <li class="list-group-item"><ul><li>Update entries</li></ul>
38 ></li>
39             <li class="list-group-item"><ul><li>Delete entries</li></ul>
40 ></li>
41         </ul>
42     </div>
43 </div>
44 <div class = "col">
45     <div class="card" style="width: 20rem;">
46         <div class="card-body">
47             <h5 class="card-title">Get Started</h5>
48             <p class="card-text">To get started, click one of the tabs
49 on the navigation bar above, that's all!</p>
50         </div>
51     </div>
52 <br/>
53     
55 </div>
56 </div>
57 </div>

```

2.4.3 Ship Technics

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

```

1 def get_ship_d1():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             """SELECT * FROM ship_details_1
7             JOIN (SELECT ship_id, name FROM ships) AS ship_names
8             ON ship_details_1.ship_id = ship_names.ship_id"""
9         ).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:

```
1 class ShipD1Form(FlaskForm):
2     ship_id = StringField()
3     model = StringField()
4     primary_role = StringField()
5     secondary_role = StringField()
6     imo = IntegerField()
7     mmsi = IntegerField()
8     abs = IntegerField()
```

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

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

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

```
1 def add_ship_d1(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new ship_d1
7         ship_d1_columns = cur.execute("PRAGMA table_info(ship_details_1)").
8         fetchall()
9
10        new_ship_d1 = []
11        for column in ship_d1_columns:
12            if column["name"] in request.form.keys():
13                new_ship_d1 += [request.form[column["name"]]]
14
15        cur.execute(f'INSERT INTO ship_details_1 VALUES ({",".join("? " *
16        len(ship_d1_columns))})', new_ship_d1)
17        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:

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

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

```
1 def update_ship_d1(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Update ship_d1
7         ship_d1_columns = cur.execute("PRAGMA table_info(ship_details_1)").
8         fetchall()
9
10        ship_d1_columns_str = ",".join([column["name"] + "=? " for column in
11        ship_d1_columns if column["name"] != "ship_id"])
12
13        ship_d1 = []
14        for column in ship_d1_columns:
```

```

13         if column["name"] in request.form.keys():
14             ship_d1 += [request.form[column["name"]]]
15
16         print("SHIP D1 COL STR: " + ship_d1_columns_str)
17         ship_id = ship_d1[0]
18         ship_d1 = ship_d1[1:] + [ship_id]
19
20         cur.execute(f'UPDATE ship_details_1 SET {ship_d1_columns_str} WHERE
ship_id = ?', ship_d1)
21         con.commit()

```

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

```

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

```

SQL statement for deleting Technics information in database.py:

```

1 def delete_ship_d1(ship_id):
2     with sqlite3.connect(db_location) as con:
3         cursor = con.cursor()
4         query = "DELETE FROM ship_details_1 WHERE (ship_id = ?)"
5         cursor.execute(query, (ship_id,))
6         con.commit()

```

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

```

1 function delete_ship_d1(ship_id) {
2     document.location = '/delete_ship_detail_1?ship_id=' + ship_id
3 }

```

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:

```

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

```

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

```

1 class PayloadForm(FlaskForm):
2     payload_id = StringField()
3     name = StringField()
4     type = StringField()
5     reused = SelectField(choices=[("True", "Yes"), ("False", "No")])
6     manufacturers = StringField()
7     mass_kg = IntegerField()
8     mass_lb = IntegerField()
9     orbit = StringField()
10    reference_system = StringField()
11    regime = StringField()

```

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

```

1 def get_payloads():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row

```

```

4         cur = con.cursor()
5         return cur.execute(
6             "SELECT * FROM payloads ORDER BY name ASC"
7         ).fetchall()

```

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

```

1 <table class = "table table-hover table-sm">
2     <thead>
3         <tr class = "table-active">
4             <th>Name</th>
5             <th>Type</th>
6             <th>Is Reused</th>
7             <th>Manufacturers</th>
8             <th>Mass (kg)</th>
9             <th>Mass (lb)</th>
10            <th>Orbit</th>
11            <th>Reference System</th>
12            <th>Regime</th>
13            <th colspan="2">Editing</th>
14        </tr>
15    </thead>
16    <tbody>
17        {% for payload in payloads %}
18        <tr class>
19            <td>
20                {{payload["name"]}}
21            </td>
22            <td>
23                {{payload["type"]}}
24            </td>
25            <td>
26                {{"No" if payload["reused"] == "False" else "Yes"}}
27            </td>
28            <td>
29                {{ "-" if payload["manufacturers"] == None else payload["
manufacturers"]}}
30            </td>
31            <td>
32                {{ "-" if payload["mass_kg"] == None else payload["mass_kg"
]]}}
33            </td>
34            <td>
35                {{ "-" if payload["mass_lb"] == None else payload["mass_lb"
]]}}
36            </td>
37            <td>
38                {{ "-" if payload["orbit"] == None else payload["orbit"]}}
39            </td>
40            <td>
41                {{ "-" if payload["reference_system"] == None else payload["
reference_system"]}}
42            </td>
43            <td>
44                {{ "-" if payload["regime"] == None else payload["regime"]}}
45            </td>
46            <td>
47                <button type="button" class="btn btn-warning btn-sm" data-
toggle="modal" data-target="#update-payload-modal-{{ payload.payload_id
}}">
48                    Update
49                </button>

```

```

50         </td>
51         <td>
52             {% if current_user.is_admin %}
53             <button type="button" class="btn btn-danger btn-sm" data-
toggle="modal"
54                 onclick="delete_payload('{{ payload.payload_id }}')">
55                 Delete
56             </button>
57             {% endif %}
58         </td>
59     </tr>
60     {% endfor %}
61 </tbody>
62 </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:

```

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

```

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

```

1 <div class="modal fade" id="add-payload-modal" tabindex="-1" role="dialog"
aria-labelledby="add-payload-modal-label" aria-hidden="true">
2     <div class="modal-dialog" role="document">
3         <div class="modal-content">
4             <div class="modal-header">
5                 <h5 class="modal-title" id="add-payload-modal-label">Add
Payload</h5>
6                 <button type="button" class="btn-close" data-dismiss="modal"
aria-label="Close"></button>
7             </div>
8             <div class="modal-body"><form action="/add_payload" method="
POST" style="display: grid;">
9                 {{ formM.csrf_token }}
10                 <div class="field">
11                     <label for="name" class="form-label detail-field">Name :
</label>
12                     <div class="control detail-field">
13                         {{ formM.name(class="form-control-sm") }}
14                     </div>
15                 </div>
16                 <div class="field">
17                     <label for="type" class="form-label detail-field">Type :
</label>
18                     <div class="control detail-field">
19                         {{ formM.type(class="form-control-sm") }}
20                     </div>
21                 </div>
22                 <div class="field">
23                     <label for="reused" class="form-label detail-field">
Reused?:</label>
24                     <div class="control detail-field">
25                         {{ formM.reused(class="form-control-sm") }}
26                     </div>
27                 </div>
28                 <div class="field">

```



```

29         <label for="manufacturers" class="form-label detail-
field">Manufacturers:</label>
30         <div class="control detail-field">
31             {{ formM.manufacturers(class="form-control-sm") }}
32         </div>
33     </div>
34     <div class="field">
35         <label for="mass_kg" class="form-label detail-field">
Mass (kg):</label>
36         <div class="control detail-field">
37             {{ formM.mass_kg(class="form-control-sm") }}
38         </div>
39     </div>
40     <div class="field">
41         <label for="mass_lb" class="form-label detail-field">
Mass (lb):</label>
42         <div class="control detail-field">
43             {{ formM.mass_lb(class="form-control-sm") }}
44         </div>
45     </div>
46     <div class="field">
47         <label for="orbit" class="form-label detail-field">
Orbit:</label>
48         <div class="control detail-field">
49             {{ formM.orbit(class="form-control-sm") }}
50         </div>
51     </div>
52     <div class="field">
53         <label for="reference_system" class="form-label detail-
field">Reference System:</label>
54         <div class="control detail-field">
55             {{ formM.reference_system(class="form-control-sm")
}}
56         </div>
57     </div>
58     <div class="field">
59         <label for="regime" class="form-label detail-field">
Regime:</label>
60         <div class="control detail-field">
61             {{ formM.regime(class="form-control-sm") }}
62         </div>
63     </div>
64
65     <button type="submit" class="btn btn-warning">Add</button>
66 </form>
67 </div>
68 <div class="modal-footer">
69     <button type="button" class="btn btn-secondary" data-
dismiss="modal">Close</button>
70 </div>
71 </div>
72 </div>
73 </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:

```

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

```

```
5 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:

```
1 def add_payload(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new payload
7         payload_columns = cur.execute("PRAGMA table_info(payloads)").
            fetchall()
8
9         new_payload = [str(current_app.config["payload_id"])] # Get next
            ID
10        current_app.config["payload_id"] += 1
11        for column in payload_columns:
12            if column["name"] in request.form.keys():
13                new_payload += [request.form[column["name"]]]
14
15        cur.execute(f'INSERT INTO payloads VALUES ({",".join("?" * len(
            payload_columns))})', new_payload)
16        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.

```
1 {% for payload in payloads %}
2
3 <div class="modal fade" id="update-payload-modal-{{ payload.payload_id }}"
4     tabindex="-1" role="dialog" aria-labelledby="update-payload-modal-label-
5     -{{ payload.payload_id }}" aria-hidden="true">
6     <div class="modal-dialog" role="document">
7         <div class="modal-content">
8             <div class="modal-header">
9                 <h5 class="modal-title" id="payload-modal-label-{{ payload.
10                 payload_id }}"> Update {{ payload.name }} </h5>
11             </div>
12             <div class="modal-body"><form action="/update_payload" method="POST"
13             " style="display: grid;">
14                 {{ formM.csrf_token }}
15                 <div class="field" style="display: none">
16                     <div class="control">
17                         {{ formM.payload_id(value=payload.payload_id, class="form
18                         -control-sm") }}
19                     </div>
20                 </div>
21                 <div class="field">
22                     <label for="name" class="form-label detail-field">Name:</
23                     label>
24                     <div class="control detail-field">
25                         {{ formM.name(value=payload.name, class="form-control-sm")
26                     }}
27                 </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">

```

```

68         <label for="reference_system" class="form-label detail-field">
Reference System:</label>
69         <div class="control detail-field">
70             {{ formM.reference_system(value=payload.reference_system,
class="form-control-sm") }}
71         </div>
72     </div>
73     <div class="field">
74         <label for="regime" class="form-label detail-field">Regime:</
label>
75         <div class="control detail-field">
76             {{ formM.regime(value=payload.regime,class="form-control-sm
") }}
77         </div>
78     </div>
79
80     <button type="submit" class="btn btn-warning">Update</button>
81 </form>
82 </div>
83 <div class="modal-footer">
84     <button type="button" class="btn btn-secondary" data-dismiss="
modal">Close</button>
85 </div>
86 </div>
87 </div>
88 </div>
89 {% endfor %}

```

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

```

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

```

The database operations are as follows in database.py:

```

1 def update_payload(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Update payload
7         payload_columns = cur.execute("PRAGMA table_info(payloads)").
fetchall()
8         payload_columns_str = ",".join([column["name"] + "=? " for column in
payload_columns if column["name"] != "payload_id"])
9
10        payload = []
11        for column in payload_columns:
12            if column["name"] in request.form.keys():
13                payload += [request.form[column["name"]]]
14
15        payload_id = payload[0]
16        payload = payload[1:] + [payload_id]
17
18        cur.execute(f'UPDATE payloads SET {payload_columns_str} WHERE
payload_id = ?', payload)
19        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:

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

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

```
1 def delete_payload(payload_id):
2     with sqlite3.connect(db_location) as con:
3         cursor = con.cursor()
4         print("USER IS ADMIN, DELETE PAYLOAD")
5         #cursor.execute("PRAGMA foreign_keys=ON") # This allows for
6         cascading to details
7         query = "DELETE FROM payloads WHERE (payload_id = ?)"
8         cursor.execute(query, (payload_id,))
9         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:

```
1 <button type="button" class="btn btn-warning" data-toggle="modal" data-
   target="#filter-payload-modal" style="position: fixed; bottom: 2.5em;
   right: 2.5em;">
2     Search in Payloads
3 </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:

```
1 <div class="modal fade" id="filter-payload-modal" tabindex="-1" aria-
   labelledby="filter-payload-modal-label" aria-hidden="true">
2     <div class="modal-dialog modal-sm">
3         <div class="modal-content">
4             <div class="modal-header">
5                 <h5 class="modal-title" id="filter-payload-modal-label">
6                     Search</h5>
7                 <button type="button" class="btn-close" data-dismiss="modal"
8                     aria-label="Close"></button>
9             </div>
10            <div class="modal-body"><form action="/payloads_filtered"
11                method="POST">
12                <label for="fname" class="form-label">Name:</label>
13                <input type="text" class="form-control" name="fname">
14
15                <label for="ftype" class="form-label">Type:</label>
16                <input type="text" class="form-control" name="ftype">
```

```

14
15         <label for="freused" class="form-label">Reused:</label>
16         <select class="form-select" name="freused" aria-label="
Default Select">
17             <option value=""> Any </option>
18             <option value="True"> Yes </option>
19             <option value="False"> No </option>
20         </select>
21
22         <label for="fmanufacturers" class="form-label">
Manufacturers:</label>
23         <input type="text" class="form-control" name="
fmanufacturers">
24
25         <label for="forbit" class="form-label">Orbit:</label>
26         <input type="text" class="form-control" name="forbit">
27
28         <label for="reference_system" class="form-label">
Reference System:</label>
29         <input type="text" class="form-control" name="
reference_system">
30
31         <label for="fregime" class="form-label">Regime:</label>
32         <input type="text" class="form-control" name="fregime">
33
34         <div class="modal-footer">
35             <button type="button" class="btn btn-secondary"
data-dismiss="modal">Close</button>
36             <button type="submit" class = "btn btn-warning">
Search</button>
37         </div>
38     </form>
39 </div>
40 </div>
41 </div>
42 </div>

```

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

```

1 @views.route('/payloads_filtered', methods=['GET', 'POST'])
2 def payloads_filtered():
3     filter_data = database.filter_payloads(request)
4     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:

```

1 def filter_payloads(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         print("filter_payloads executed")
6         query = "SELECT * FROM payloads"
7         params = []
8         print(request.form.get('fname'))
9         if request.form.get('fname'):
10             if not params:
11                 query += " WHERE"
12                 user_name = request.form['fname']
13                 query += " name LIKE ?"
14                 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 + "%"

```

```
75         params.append(param_regime)
76
77     print(query)
78     print(tuple(params))
79     filter = cur.execute(query, tuple(params)).fetchall()
80     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:

```
1 @views.route('/ships', methods=['GET', 'POST'])
2 def ships():
3     ship_data = database.get_ships()
4     ship_d1_data = database.get_ship_d1()
5     ship_d2_data = database.get_ship_d2()
6
7     inexistent_ship_d1 = []
8     inexistent_ship_d2 = []
9     inexistent_ship_d1_dict = []
10    inexistent_ship_d2_dict = []
11
12    present = False
13    for ship in ship_data:
14        for ship_d1 in ship_d1_data:
15            if ship["ship_id"] == ship_d1["ship_id"]:
16                present = True
17                break
18        if not present:
19            inexistent_ship_d1 += [ship["ship_id"]]
20            inexistent_ship_d1_dict += [{"ship_id": ship["ship_id"], "name"
: ship["name"]}]]
21        present = False
22
23        for ship_d2 in ship_d2_data:
24            if ship["ship_id"] == ship_d2["ship_id"]:
25                present = True
26                break
27        if not present:
28            inexistent_ship_d2 += [ship["ship_id"]]
29            inexistent_ship_d2_dict += [{"ship_id": ship["ship_id"], "name"
: ship["name"]}]]
30        present = False
31
32    return render_template("ships.html", ships=ship_data,
33        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,
34        formM=forms.ShipForm(), formD1=forms.ShipD1Form(), formD2=forms.
ShipD2Form())
```

SQL statement in database.py:

```
1 def get_ships_only():
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         return cur.execute(
6             "SELECT * FROM ships"
7             ).fetchall()
```

Jinja template for ships.html:

```
1 <table class = "table table-hover table-sm">
2     <thead>
3         <tr class = "table-active">
4             <th>Name</th>
5             <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" }}
17     </td>
18     <td>
19         {% if ship["ship_id"] in inexistent_d1 %}
20             <button type="button" class="btn btn-warning btn-sm" data-
21 toggle="modal" data-target="#add-ship-details-1-modal-{{ ship.ship_id }}">
22                 Add Technics
23             </button>
24             {% else %}
25                 <button type="button" class="btn btn-warning btn-sm" data-
26 toggle="modal" data-target="#ship-details-1-modal-{{ ship.ship_id }}">
27                     Technics
28                 </button>
29             {% endif %}
30         </td>
31     <td>
32         {% if ship["ship_id"] in inexistent_d2 %}
33             <button type="button" class="btn btn-warning btn-sm" data-
34 toggle="modal" data-target="#add-ship-details-2-modal-{{ ship.ship_id }}">
35                 Add Measurements
36             </button>
37             {% else %}
38                 <button type="button" class="btn btn-warning btn-sm" data-
39 toggle="modal" data-target="#ship-details-2-modal-{{ ship.ship_id }}">
40                     Measurements
41                 </button>
42             {% endif %}
43         </td>
44     <td>
45         <button type="button" class="btn btn-warning btn-sm" data-
46 toggle="modal" data-target="#update-ship-modal-{{ ship.ship_id }}">
47             Update
48         </button>
49     </td>
50     <td>
51         {% if current_user.is_admin %}
52             <button type="button" class="btn btn-danger btn-sm" data-
53 toggle="modal"
54             onclick="delete_ship('{{ ship.ship_id }}')">
55                 Delete
56             </button>
57         {% endif %}
58     </td>
59 </tr>
60 {% endfor %}
61 </tbody>

```

57 </table>

2.5.2 Ship Adding

The view function to add ship in views.py:

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

SQL statement for add ship in database.py:

```
1 def add_ship(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new ship
7         ship_columns = cur.execute("PRAGMA table_info(ships)").fetchall()
8
9         new_ship = [str(current_app.config["ship_id"])] # Get next ID
10        current_app.config["ship_id"] += 1
11        for column in ship_columns:
12            if column["name"] in request.form.keys():
13                new_ship += [request.form[column["name"]]]
14
15        cur.execute(f'INSERT INTO ships VALUES ({",".join("'" * len(
16            ship_columns)))', new_ship)
17        con.commit()
```

Button for adding new ship in ships.html:

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

Jinja template for adding ships.html:

```
1 <div class="modal fade" id="add-ship-modal" tabindex="-1" role="dialog"
    aria-labelledby="add-ship-modal-label" aria-hidden="true">
2     <div class="modal-dialog" role="document">
3         <div class="modal-content">
4             <div class="modal-header">
5                 <h5 class="modal-title" id="ship-modal-label"> Add Ship </h5>
6             </div>
7             <div class="modal-body"><form action="/add_ship" method="POST" style=
                "display: grid;">
8                 {{ formM.csrf_token }}
9                 <div class="field">
10                     <label for="name" class="form-label detail-field">Name:</label>
11                     <div class="control detail-field">
12                         {{ formM.name(class="form-control-sm") }}
13                     </div>
14                 </div>
15                 <div class="field">
16                     <label for="type" class="form-label detail-field">Type:</label>
17                     <div class="control detail-field">
18                         {{ formM.type(class="form-control-sm") }}
19                     </div>
20                 </div>
```

```

21     <div class="field">
22         <label for="active" class="form-label detail-field">Active:</
label>
23         <div class="control detail-field">
24             {{ formM.active(class="form-control-sm") }}
25         </div>
26     </div>
27     <button type=submit class="btn btn-warning">Add</button>
28 </form>
29 </div>
30 <div class="modal-footer">
31     <button type="button" class="btn btn-secondary" data-dismiss="modal
">Close</button>
32 </div>
33 </div>
34 </div>
35 </div>

```

2.5.3 Ship Deleting

The view function to delete ship in views.py:

```

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

```

SQL statement for deleting ship in database.py:

```
1 cursor = con.cursor()
2 cursor.execute("PRAGMA foreign_keys=ON") # This allows for
cascading to details
3 query = "DELETE FROM ships WHERE (ship_id = ?)"
4 cursor.execute(query, (ship_id,))
5 con.commit()
6
7 def delete_ship_d1(ship_id):
```

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

```
1 function delete_ship(ship_id) {
2     document.location = '/delete_ship?ship_id=' + ship_id
3 }
```

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.

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

The view function to update ship in views.py:

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

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

```
1 con.row_factory = sqlite3.Row
2 cur = con.cursor()
3
4 # Update ship
5 ship_columns = cur.execute("PRAGMA table_info(ships)").fetchall()
6 ship_columns_str = ",".join([column["name"] + "=? " for column in
ship_columns if column["name"] != "ship_id"])
7
8 ship = []
9 for column in ship_columns:
10     if column["name"] in request.form.keys():
11         ship += [request.form[column["name"]]]
12
13 ship_id = ship[0]
14 ship = ship[1:] + [ship_id]
15
16 cur.execute(f'UPDATE ships SET {ship_columns_str} WHERE ship_id = ?
', ship)
17 con.commit()
18
19 def update_ship_d1(request):
```

2.5.5 Ships Filtering

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

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

The view function to filter ship in views.py:

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

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

```
1 def filter_ships(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5         print("filter_ships executed")
6         query = "SELECT * FROM ships"
7         params = []
8         print(request.form.get('fname'))
9         if request.form.get('fname'):
10             if not params:
11                 query += " WHERE"
12                 user_name = request.form['fname']
13                 query += " name LIKE ?"
14                 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('factive') != "":
28             if not params:
29                 query += " WHERE"
30             else:
31                 query += " AND"
32                 user_active = request.form['factive']
33                 query += " active LIKE ?"
34                 param_active = "%" + user_active + "%"
35                 params.append(param_active)
36
37         print(query)
38         print(tuple(params))
39         filter = cur.execute(query, tuple(params)).fetchall()
40         return filter
```

Button for filter ships in ships.html:

```

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

```

Jinja template for filter ships.html:

```

1 <div class="modal fade" id="filter-ship-modal" tabindex="-1" aria-
  labelledby="filter-ship-modal-label" aria-hidden="true">
2   <div class="modal-dialog modal-sm">
3     <div class="modal-content">
4       <div class="modal-header">
5         <h5 class="modal-title" id="filter-ship-modal-label">Search</
  h5>
6         <button type="button" class="btn-close" data-dismiss="modal"
  aria-label="Close"></button>
7       </div>
8       <div class="modal-body"><form action="/ships_filtered" method="
  POST">
9         <label for="fname" class="form-label">Name:</label>
10        <input type="text" class="form-control" name="fname">
11
12        <label for="ftype" class="form-label">Type:</label>
13        <input type="text" class="form-control" name="ftype">
14
15        <label for="factive" class="form-label">Active:</label>
16        <select class="form-select" name="factive" aria-label="
  Default Select">
17          <option value=""> Any </option>
18          <option value="True"> Active </option>
19          <option value="False"> Deactive </option>
20        </select>
21
22        <div class="modal-footer">
23          <button type="button" class="btn btn-secondary" data-
  dismiss="modal">Close</button>
24          <button type="submit" class = "btn btn-warning">
  Search</button>
25        </div>
26      </form>
27    </div>
28  </div>
29 </div>
30 </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:

```

1 class ShipD2Form(FlaskForm):
2     ship_id = StringField()
3     class_ = DecimalField()
4     mass_kg = IntegerField()
5     mass_lb = IntegerField()
6     year_built = IntegerField()
7     home_port = StringField()

```

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

```

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

```

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

```

1 def add_ship_d2(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Add new ship_d2
7         ship_d2_columns = cur.execute("PRAGMA table_info(ship_details_2)").
fetchall()
8
9         new_ship_d2 = []
10        for column in ship_d2_columns:
11            if column["name"] in request.form.keys():
12                new_ship_d2 += [request.form[column["name"]]]
13
14        cur.execute(f'INSERT INTO ship_details_2 VALUES ({",".join("? " *
len(ship_d2_columns))})', new_ship_d2)
15        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.

```

1 class ShipD2Form(FlaskForm):
2     ship_id = StringField()
3     class_ = DecimalField()
4     mass_kg = IntegerField()
5     mass_lb = IntegerField()
6     year_built = IntegerField()
7     home_port = StringField()

```

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

```

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

```

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

```

1 def update_ship_d2(request):
2     with sqlite3.connect(db_location) as con:
3         con.row_factory = sqlite3.Row
4         cur = con.cursor()
5
6         # Update ship_d2
7         ship_d2_columns = cur.execute("PRAGMA table_info(ship_details_2)").
fetchall()
8         ship_d2_columns_str = ",".join([column["name"] + "=? " for column in
ship_d2_columns if column["name"] != "ship_id"])
9
10        ship_d2 = []
11        for column in ship_d2_columns:
12            if column["name"] in request.form.keys():
13                ship_d2 += [request.form[column["name"]]]

```



```

14         ship_id = ship_d2[0]
15         ship_d2 = ship_d2[1:] + [ship_id]
16         print(ship_d2)
17
18         print("SHIP D2 COL STR: " + ship_d2_columns_str)
19         cur.execute(f'UPDATE ship_details_2 SET {ship_d2_columns_str} WHERE
20 ship_id = ?', ship_d2)
21         con.commit()

```

2.5.8 Ship Measurement Deleting

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

```

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

```

SQL statement for deleting ship measurement in database.py:

```

1 def delete_ship_d2(ship_id):
2     with sqlite3.connect(db_location) as con:
3         cursor = con.cursor()
4         query = "DELETE FROM ship_details_2 WHERE (ship_id = ?)"
5         cursor.execute(query, (ship_id,))
6         con.commit()

```

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

```

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

```

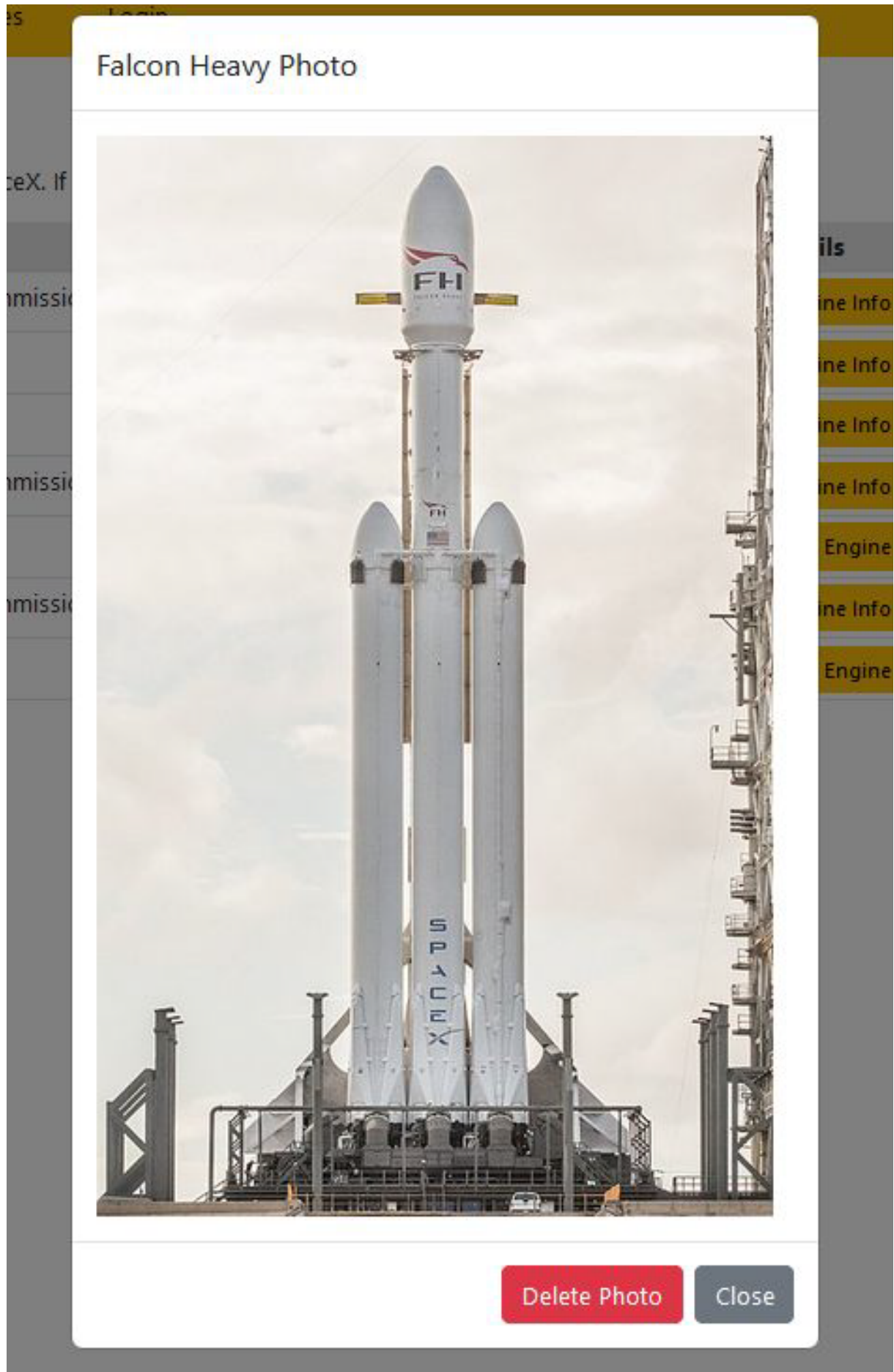


Figure 23: Rocket image view

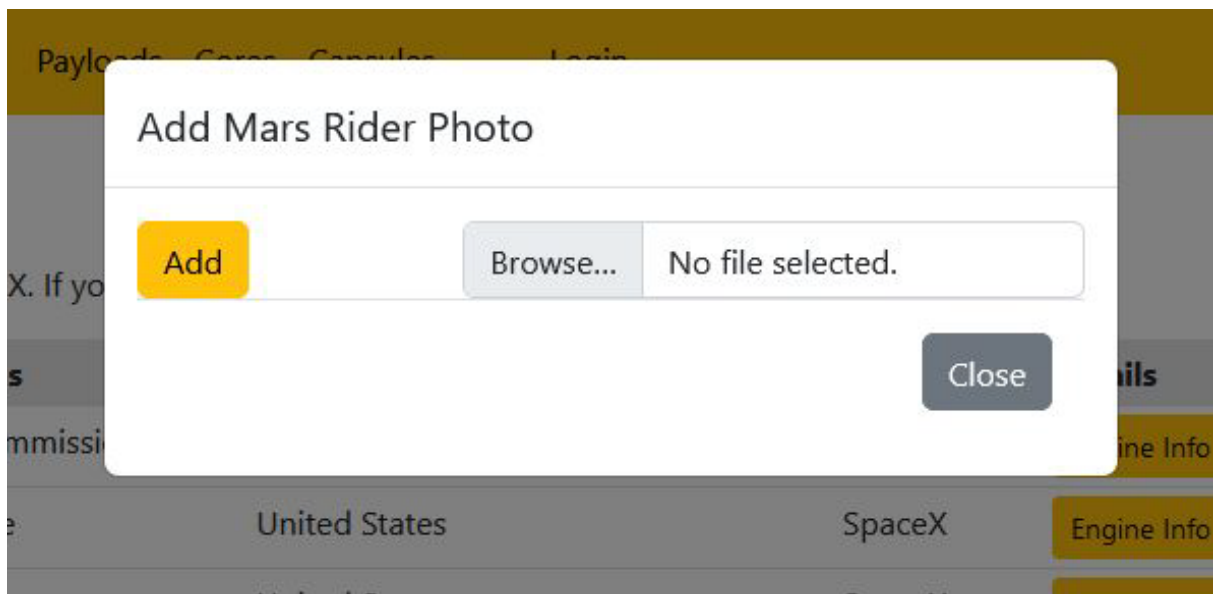


Figure 24: Rockets images adding view

Home Launches Launchpads Rockets Ships Payloads Cores Capsules You are logged in as 1234. Logout																																																															
<h2>Launchpads</h2> <p>These are SpaceX's launchpads.</p> <table> <tr> <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> <tr> <td>CCSFS SLC 40</td><td>Cape Canaveral Space Force Station Space Launch Complex 40</td><td>active</td><td>Cape Canaveral</td><td>Florida</td><td>America / New York</td><td>28.5618571</td><td>-80.577366</td></tr> <tr> <td>KSC LC 39A</td><td>Kennedy Space Center Historic Launch Complex 39A</td><td>active</td><td>Cape Canaveral</td><td>Florida</td><td>America / New York</td><td>28.6080585</td><td>-80.6039558</td></tr> <tr> <td>Kwajalein Atoll</td><td>Kwajalein Atoll Omelek Island</td><td>retired</td><td>Omelek Island</td><td>Marshall Islands</td><td>Pacific / Kwajalein</td><td>9.0477206</td><td>167.7431292</td></tr> <tr> <td>STLS</td><td>SpaceX South Texas Launch Site</td><td>under construction</td><td>Boca Chica Village</td><td>Texas</td><td>America / Chicago</td><td>25.9972641</td><td>-97.1560845</td></tr> <tr> <td>VAFB SLC 3W</td><td>Vandenberg Air Force Base Space Launch Complex 3W</td><td>retired</td><td>Vandenberg Air Force Base</td><td>California</td><td>America / Los Angeles</td><td>34.6440904</td><td>-120.5931438</td></tr> <tr> <td>VAFB SLC 4E</td><td>Vandenberg Air Force Base Space Launch Complex 4E</td><td>active</td><td>Vandenberg Air Force Base</td><td>California</td><td>America / Los Angeles</td><td>34.632093</td><td>-120.610829</td></tr> </table>								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
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

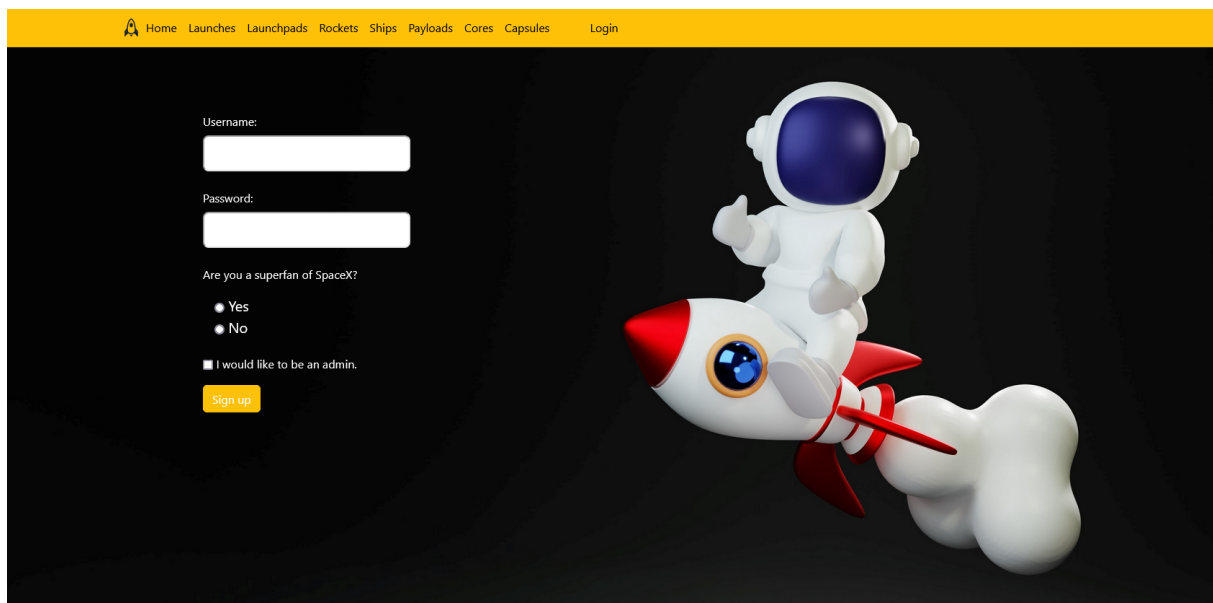


Figure 26: Signup page

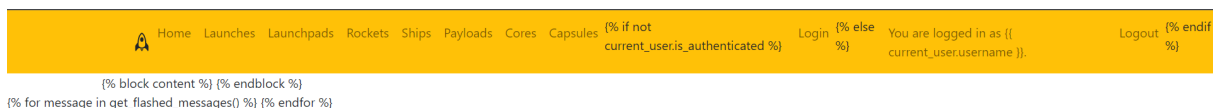


Figure 27: Base Page Design

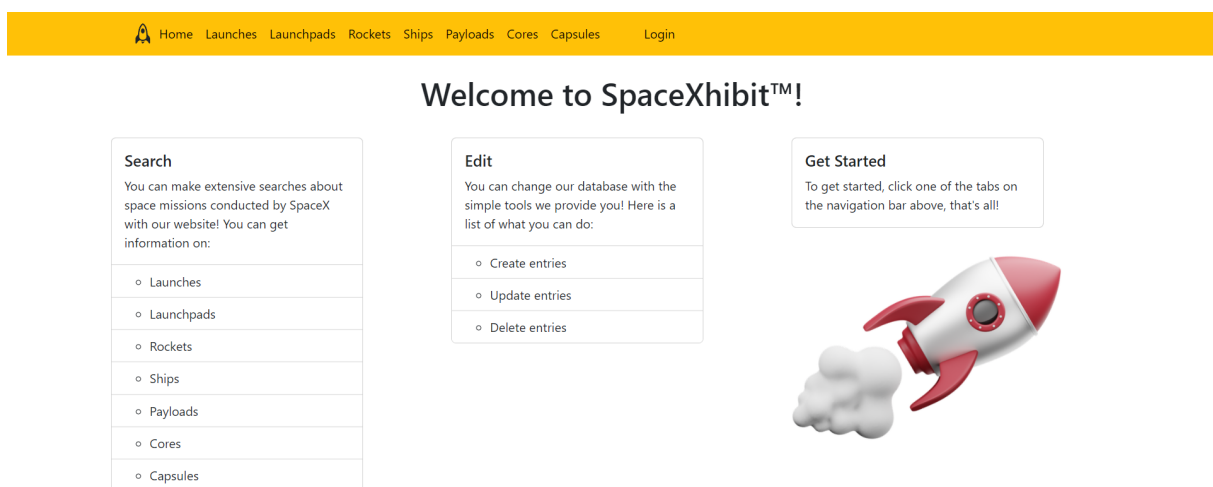


Figure 28: Home Page Design

Ships

These are the ships own and operated by SpaceX. If you see anything wrong or missing, feel free to amend. Please [login](#) to delete or ships data.

Name	Type	Status	Details	Editing
A Shortfall of Gravitas	Barge	Deactive	Technics	Measurements Update

Figure 29: Technics Button on Ships Page

A Shortfall of Gravitas Technical Info

Model	-
Primary Role:	ASDS barge
Secondary Role:	-
IMO:	-
MMSI	-
ABS:	-

[Delete Technical Info](#)[Update](#)[Close](#)

Figure 30: Technics Modal

Update A Shortfall of Gravitas Technical Info

model:

-

Primary Role:

ASDS barge

Secondary Role:

-

imo:

mmsi:

abs:

Update

Close

Figure 31: Technics Update

Home Launches Launchpads Rockets Ships Payloads Cores Capsules Login									
<h2>Payloads</h2> <p>These are the payloads that has been sent to space by SpaceX Rockets so far. If you see anything wrong or missing, feel free to amend. Please login to update, delete or add payload data.</p>									
Name	Type	Is Reused	Manufacturers	Mass (kg)	Mass (lb)	Reference Orbit	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	Update
Beresheet	Lander	No	SSL	585.0	1289.7	GTO	geocentric	highly-elliptical	New Payload Search in Payloads

Figure 32: Payloads Page

Add Payload

×

Name:

Type:

Reused?:

Yes ▾

Manufacturers:

Mass (kg):

Mass (lb):

Orbit:

Reference System:

Regime:

Add

Close

Figure 33: Add Payload Form

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

Figure 34: Delete Payload Button

Update ABS-2A

Name:

ABS-2A

Type:

Satellite

Reused?:

No ▾

Manufacturers:

Boeing

Mass (kg):

1800,0

Mass (lb):

3950,0

Orbit:

GTO

Reference System:

geocentric

Regime:

geostationary

Update

Close

Figure 35: Update Payload Form

Search

×

Name:

Type:

Reused:

Any

▼

Manufacturers:

Orbit:

Reference System:

Regime:

Close

Search

Figure 36: Search Payload Form

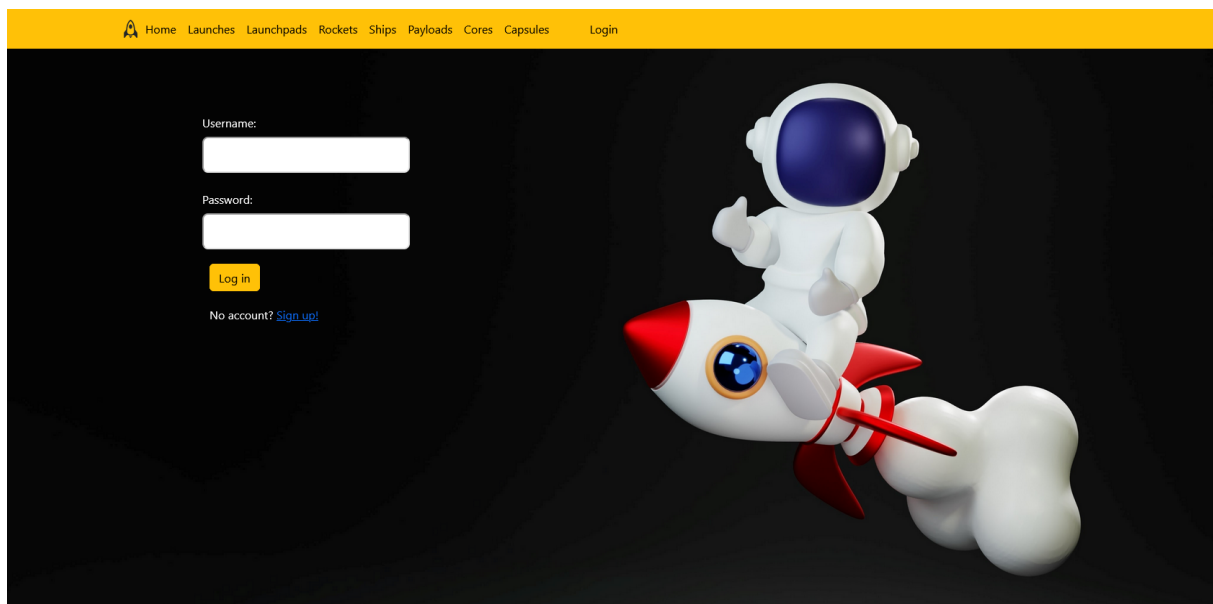


Figure 37: Login Page

Home Launches Launchpads Rockets Ships Payloads Cores Capsules You are logged in as admin. Logout

Ships

These are the ships own and operated by SpaceX. If you see anything wrong or missing, feel free to amend.

Name	Type	Status	Details	Editing
A Shortfall of Gravitas	Barge	Deactive	Technics Measurements	Update Delete
American Champion	Tug	Deactive	Technics Measurements	Update Delete
American Islander	Cargo	Deactive	Technics Measurements	Update Delete
American Spirit	Cargo	Deactive	Technics Measurements	Update Delete
Betty R Gambarella	Tug	Deactive	Technics Measurements	Update Delete
Elsbeth III	Tug	Deactive	Technics Measurements	Update Delete
Finn Falgout	Tug	Active	Technics Measurements	Update Delete
GO Ms Chief	High Speed Craft	Active	Technics Measurements	Update Delete
GO Ms Tree	High Speed Craft	Active	Technics Measurements	Update Delete
GO Navigator	Cargo	Active	Technics Measurements	Update Delete
GO Pursuit	Cargo	Deactive	Technics Measurements	Update Delete
GO Quest	Cargo	Active	Technics Measurements	Update Delete
GO Searcher	Cargo	Active	Technics Measurements	Update Delete
HAWK	Tug	Deactive	Technics Measurements	Update Delete
Hollywood	Tug	Active	Technics Measurements	Update Delete
Just Read The Instructions 1	Barge	Deactive	Technics Measurements	Update Delete
Just Read The Instructions 2	Barge	Active	Technics Measurements	Update Delete
Kelly C	Tug	Deactive	Technics Measurements	Update Delete

New Ship
Search in Ships

Figure 38: Ships Page

Add Ship

Name:

Type:

Active:

Active ▾

Add

Close

Figure 39: Add ship Form

Name	Type	Status	Details		Editing	
A Shortfall of Gravitas	Barge	Deactive	Technics	Measurements	Update	Delete

Figure 40: Delete ship Button

Update A Shortfall of Gravitas

Name:

Type:

Active:


Deactive ▾

Update

Close

Figure 41: Update ship Form


Search



Name:

Type:

Active:

Any

Close

Search

Figure 42: Search ship Form

GO Pursuit Measurements

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



Figure 43: Measurement Modal



Figure 44: Add measurement Button

Add ForTheReport Measurements

Class:

Year Built:

Home Port:

Mass (kg):

Mass (lb):

Add

Close

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:	<input type="text" value="7427463,0"/>
Mass (kg):	<input type="text" value="202302,0"/>
Mass (lb):	<input type="text" value="446000,0"/>
Year Built:	<input type="text" value="1974,0"/>
Home Port:	<input type="text" value="Port of Los Angeles"/>

Update

Close

Figure 47: Update Measurement Form