Musa Rafik - mar6827
Lucas Best - lwb498
Jack Diao - qd572
Shawn Victor - sfv225
Kenan Hurd - kah4285

EE461L Phase 3 Report

URL's

Site: http://d29srdcmzi6q02.cloudfront.net/
Github: https://github.com/qinyudiao/Software-Desgin-Lab-team-project

Team and Project Information:

Project Name: Every Rocket Launch
Team Canvas Group: morning-8

Name	Email	Github Username
Jack Diao	qdiao@utexas.edu	qinyudiao
Musa Rafik	musa_rafik@utexas.edu	musarafik
Lucas Best	lucasbest@utexas.edu	LBest42
Shawn Victor	shawnfvictor@gmail.com	shawnvictor
Kenan Hurd	hurdkenan@gmail.com	KenaHu

Motivation and Users

There has been a resurgence in the interest in space exploration in the past couple of years. Several companies have been developed with the hopes of exploring the new frontier, such as SpaceX and Blue Origin. As a result, there is an abundance of information related to the topic, but no centralized location to view all data. The goal of Every Rocket Launch is to be that centralized location. Through the web application, users will be able to view different launches and space companies looking to expand to space.

Every Rocket Launch is intended for anyone, from a space novice to even researchers as the web application will be pulling data from various sources. Thus, users will be able to view things like how much rocket costs, or which astronauts went on which launches. In short, anyone will be able to visit Every Rocket Launch and learn something new about space.

Requirements

User Stories

Story	Phase	Implementation on Website	Estimated Time to Completion	Actual Time to Completion
As a user, I would like to be able to click on an instance page for an agency to learn more about the agency.	3	Instance pages for agencies have a link to Wikipedia if a page exists for that agency.	5 hours	7 hours
As a researcher, I would like to be able to click on a link on an individual page of the astronauts for further research.	3	Instance pages for astronauts have a link to Wikipedia if the page exists for the astronaut.	5 hours	12 hours
As a user, I would like to click on an agency and see information about it.	3	Clicking on an agency in the model page takes the user to an instance page with information about the agency.	3 hours	5 hours
As a lazy user, I want to see basic information pop up when I hover over a specific icon of a launch pad.	3	Map displays information when the user hovers over the icons.	10 hours	8 hours
As an impatient user, I would like to look at an image while waiting for the information to load.	3	The models pages have background images while loading	2 hours	3 hours
As someone who hates getting emails from companies, I would like to be able enter my email	2	Unsubscribe form on the landing page	3 hours	5 hours

and unsubscribe from getting weekly launch updates if I have already subscribed.				
As a user who clicks on some links that are not valid anymore, I want the PageNotFound page to have something that can be clicked to redirect back to valid pages and/or the page automatically redirects back to the home page after a certain period of time.	2	PageNotFound page when directed to an invalid link	2 hours	8 hours
As a user, I want to see a sorted list of all the countries so that I can see which countries have ever launched a rocket and how many they have launched.	2	Country list component on Map page	4 hours	6 hours
As a user, I want a map function, so that I can find where each launchpad that has launched a rocket is at.	2	Everything else on the Map page	8 hours	20 hours
As a user, I want to go to a page which lists the next 10 scheduled so that I can see all the information of each launch including a countdown timer.	2	Upcoming Launches page	8 hours	14 hours
As someone who wants to stay up to date on the news, I want to be able to enter my email and receive information about the next couple of launches each week.	2	Subscribe form on landing page and each Sunday email is sent out using cron job	5 hours	12 hours
As a user, I would like to be able to go onto the home page and see when the next rocket launch will occur.	2	Text at top of landing page shows next launch to occur and description about it	3 hours	6 hours
As a user, I want to be able to see all rocket launches.	2	Launch page displays all launches and allows user to	3 hours	5 hours

		page through them		
As a user, I want to be able to view a video of failed rocket launches on an embedded video player.	1	Video embedded on Failed page	30 minutes	1 hour
As someone interested in learning about rocket launches, I want to be able to click on different tabs that will take me to pages about companies, countries, astronauts, etc.	1	Navigation bar allows users to navigate to companies, astronauts, launches, etc.	1 hour	3 hours
As an uninformed user, I would like to be able to visit a page that has information/links about rockets.	1	Links included on Education page	30 minutes	30 minutes
As a picky user, I want an animation of a rocket launching that I can view.	1	Gifs included in carousel on landing page	1 hour	45 minutes
As a user, I want to click on a link and see who contributed to the website.	1	Navigation bar has link for About page with contributor information	2 hours	4 hours
As a curious user, I want to be able to see a short introductory film so that I can learn something about rockets.	1	Video embedded on Education page discussing rockets	30 minutes	1 hour

Customer Stories

Story	Phase	Implementation on Website	Estimated Time to Completion	Actual Time to Completion
As someone interested in different countries' space efforts, I would like to see information about how many	3	Map has a table that shows countries and how many	10 hours	15 hours

launches each country has done.		launches they have done.		
As a user, I would like to see an image of the agency or their logo when I go onto their instance page.	3	Images are pulled from Wikipedia (and stored in a database) and displayed on instance pages of agencies if available.	7 hours	7 hours
As someone interested in astronauts, I would like to be able to click on an astronaut's name and see a picture of them.	3	Instance pages for astronauts displays an image of the astronaut pulled from Wikipedia (and stored in a database) if available.	7 hours	15 hours
As someone curious about astronauts, I would like to filter astronauts into subcategories.	3	The Astronauts page provides a filter tool.	2 hours	2 hours
As a user, I would like to click on different filters to see different launch pads on the map.	3	Filter buttons on the map show different launch pads.	2 hours	5 hours
As an investor, I would like to see how much each rocket launch cost.	1	Launch costs are shown on Companies page	4 hours	4 hours
As an organized user, I would like to be able to sort the launch information in alphabetical order.	1	Links included on Education page	5 hours	3 hours
As a scientist, I would like to see what the purpose of the payload that was launched by the rocket was for.	1	Company page has a column for purpose	2 hours	3 hours
As an aerospace engineer, I would like to see what the	1	Launch page has a column for	3 hours	3 hours

determined cause of failure was for unsuccessful launches.		failure reason		
As a researcher, I would like to look up specific companies, dates, or countries to get all related information.	1	User can search on each page (Company, Launch, etc.) for specific instances	5 hours	4 hours

Design

The website directs the user to a splash page at first, which has a navigation bar, a message about an upcoming launch, subscribe/unsubscribe forms for our newsletter and a carousel of three static and animated rocket launch images. The navigation bar consists of several different redirect buttons, and a dropdown button. The search button is not fully implemented for this phase. The subscribe/unsubscribe form data is sent to the backend to be processed and stored/removed from the database.

The redirection buttons are named “Every Rocket Launch”, “About”, “Launches”, “Agencies”, “Astronauts”, “Map”, and “Upcoming Launches”. All the buttons redirect to a new page with the same navigation bar on top of the page.

“Every Rocket Launch” button redirects back to the landing page.

“Upcoming Launches” redirects to a page that displays a couple of launches that will occur soon, as well as information about each. Additionally, there are links that take the user to the location in Google Maps.

“About” button redirects to the About page, which contains some basic information about the website and the GitHub Statistics which is dynamically derived from GitHub’s API. The information is pulled from the Github API in our backend and sent to the frontend when the page is loaded.

“Map” button redirects to the Map page, which has a map displaying the locations of launchpads around the world and launch rank by country. The map itself has a checkbox filter which can be used to filter launch pads by active type, usage type, and country. If a launchpad is hovered, then the basic info of that pad is popped up, and clicking on the marker will make the infobox stay.

“Launches” button redirects to the Launch page, which presents the user all the rocket launches from the very first to the latest planned. The information is obtained by sending a request to the backend, which retrieves it from the database. Each launch has a name, launch time, fail reason if failed, and a video link if there is one. Users can search through each

attribute and/or sort them by alphabetical order. Clicking on a launch takes the user to the corresponding instance page with more information about the launch.

“Agencies” button redirects to the Agencies page, which presents a list of organizations with their purposes, headquarters, and countries. Each attribute can also be searched or sorted. The data is from spacefund.com, we convert a .csv file to a .json file, store it in the database and retrieve the information when the page is loaded. In addition, each company name will redirect to its profile page with further information about the agency.

“Astronauts” dropdown menu has three buttons for each type of astronaut: International, US and Russian. Each page displays all the astronauts and by clicking on an astronaut, the user can view more information about them. Similarly to the other models, the information is stored in the database and retrieved when the page is loaded.

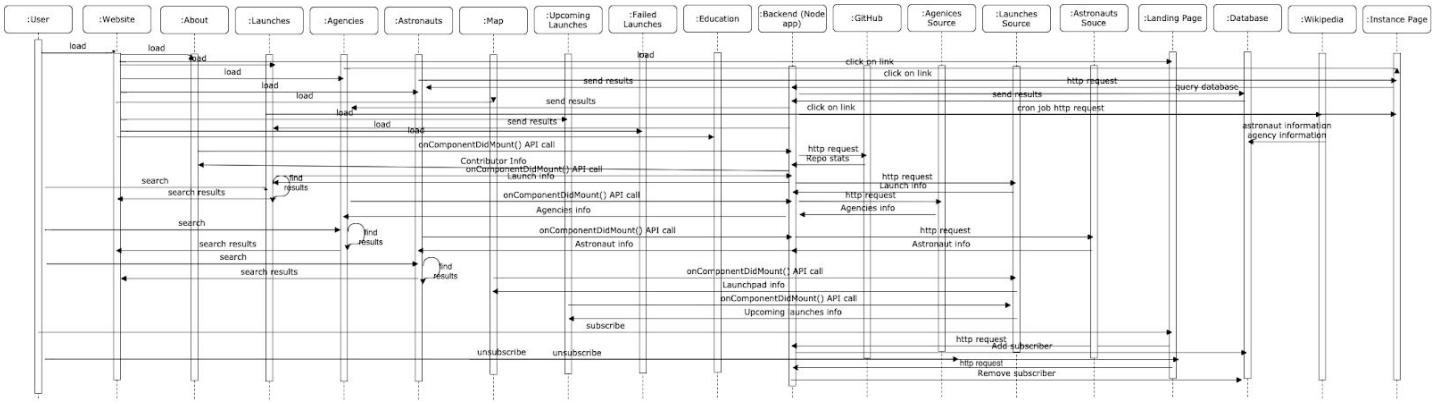
In the dropdown menu we have a “Failed_Launches” button that redirects to the Failed_Launches page, which has an embedded youtube player that plays a video of a compilation of rocket launch failures.

Another button in the menu, “Education,” redirects to the Education page, which has an embedded Youtube player that plays a short introductory film and some external links to learn more about rockets.

If directed to an invalid route, the page not found model will show up, and will redirect the user to upcoming launches pages in 5 seconds.

Use Case Diagram

Sequence Diagram



Testing

During this phase we conducted multiple tests across the frontend and backend. These tests aided in us being able to detect many bugs present in our current code base.

Backend Testing:

All backend testing was done using Mocha in combination with Chai (an assertion library) and Chai-Http (used for testing HTTP requests). For phase two, backend testing was focused on checking what was to be sent to the frontend via the many routes set-up in our main app.js file. Testing on the backend also included ensuring that functions such as subscribing and unsubscribing from the website worked as intended as well.

- Test #1-4: Checking Response sent to Frontend with Various Working Routes
    - For each of these tests, the .get function for an individual route was tested (/about, /InternationalAstronauts, /RussianAstronauts, /USAstronauts). For each of these tests, we verified that the response went through by checking the attached status as well as verified that the response was a JSON. The body of the response was also checked to ensure that there were multiple entries for each as there should be.
- Test #5: .get Response from /Launch returned only the Most Recent Launch
    - Ran the same verifications as in the previous tests for .get routes however this time we checked to ensure that only one object was being passed along.
- Test #6: Subscribe Functionality
    - Sent a POST request to the server with an email (readable by the subscriberSchema) that was already in the database to ensure that the server was properly handling such an occurrence. Checked that the response text was ‘fail’ (not an error), that the POST request went through, and that the response

was also the correct type. Console also logs that the email sent in POST is already a subscriber.

- Test #7: Unsubscribe Functionality
    - Sent a POST request to the server to unsubscribe an email that was not found in the database. Expected response text was ‘fail’ and this was verified in said test as well as checking that the request went through and that it was the proper type.

Frontend Testing:

All frontend Javascript testing was done using the react-scripts test functionality. The test suite for the frontend Javascript files consists of checking whether React components render properly using ReactDOM.

- Test #1-13: Renders without crashing
    - Takes each testable React component found in the frontend (mainly those routed out of App.js) and ensures that they each can render without crashing. This is a useful series of tests to check that React components are working as intended at their most basic levels.

GUI Testing:

All GUI testing was done using Selenium and the Chrome webdriver. The large focus on the preliminary tests written for phase two was geared around being able to detect page faults throughout our website. Due to the fact that our site has so many clickable links for more information, we wanted to make sure we knew if any redirects led to sites that would return a 404 page error.

- Test #1: Recursive Site Test for Page Errors
    - If we consider each clickable link on our site a node, and the connections between each link is the graph of all possible connections, we can use this to check all parts of our site for link errors. This test uses a recursive DFS algorithm that goes to each page, checks for clickable links, opens each link, checks inside of that, ect. After opening any link, a check is done for a page error. If a page error is found this test is not passed.
- Test #2: Launch Database YouTube Link Unavailable
    - Most of our databases contain links that have been scraped from other places. In our backend testing, we could do formatting checks on the link to detect possible bad links. However, a much more conclusive test would be to test each link redirect and check if the youtube page returns a “video unavailable” message. This test iteratively goes through all of the launch links and performs this check. If a youtube link is found to have the error, this test is not passed.

Tools, Software, Frameworks

Frontend: We used HTML, CSS, and JavaScript with React to build the front end. With React, we were able to build components that can be reused throughout our project. For example, we were able to import a navigation bar we made into all our pages with a simple command, rather than having to put together all the necessary tags one would have to use with plain HTML. In order to make the website look better, we used bootstrap and react-strap and some React components created by other people. Thanks to the modularity of these libraries, we can just import what we need, such as the carousel on the landing page. For pagination on our model pages, we used the React Table library to create a pageable table containing the information pulled from our database. We also used Moment.js to get the current time without worrying about timezones. To display a map, we used the react-google-maps module, which provides some basic google maps functionalities.

Database: We used Mongodb to store the data for our models. Due to its document-based nature we were able to easily store data of various forms.

Backend: We utilized Node with Express and Mongoose to build the backend. The advantage of this is that Node is written in JavaScript and so we are able to maintain consistency in our code. Express allows us to write endpoints for our RESTful API in a very organized and modular fashion. Mongoose allows us to communicate with MongoDB and make queries/store data. With Mongoose and Express we pull data from APIs, process the response and store it in MongoDB. Then, whenever the data needs to be rendered on the frontend, a request is sent from the frontend to our backend and the data is pulled from the database.

Hosting; We deployed our frontend to Amazon Web Services S3 and utilized Docker to make deployment easier. Docker generates our build files and then pushes them to the S3 bucket so that we do not have to do anything manually. Our backend is hosted on Amazon Web Services EC2, which is essentially a virtual machine that runs our node application. Our frontend on S3 communicates with the EC2 instance by making a request to the EC2 DNS name. Lastly, we are using Amazon Web Services Cloudfront to connect our domain name (everyrocketlaunch.com) to our S3 bucket.

Testing: We used Mocha to create unit tests for all of our JavaScript code (frontend and backend). For our GUI testing, we utilized Selenium. We were able to test the flow of our website and all of the links to ensure that they are working.

Filtering/Searching: We have a collection of cards that represent each of our models (i.e., there are cards for astronauts, launches and agencies). When the user clicks on a filter, a certain function is called that checks our collection of cards and only renders cards that meet the filter conditions. Similarly, when a user searches for a specific instance, the search term is checked against the cards and we render whichever cards match the term.

Models

Our models are agencies, launches, and astronauts. The agency model includes information about company location, purpose, and cost per launch. The launch model includes the mission name, date, status, and launch video if one exists. The astronaut model is divided by nation and includes relevant information about each astronaut listed.

We found two free databases that have information about the rocket launches that we need. The first one is https://spacefund.com/launch-database/, which provides detailed information about many aerospace agencies. We used this one to get the instances of information used in our company model. The second database is https://launchlibrary.net/docs/1.4.1/api.html which provides access to the database with an API. We used this database for our launch model instances.

For our model containing all of our Astronaut data, we used an open-source database from NASA as seen from the following link. From the model, we extracted each astronaut by country/company, name, gender, number of flights, total hours in space, etc. We plan to later develop this out to the point that the user can simply click on an astronaut name and be brought to a bio page containing more information about each astronaut.

Currently, the multimedia we have on our web application exists on the landing page, Education page, and the Failed Launch page. The video on the landing page is simply there for looks, but the videos found on the failed launches page and education page serve to inform viewers of the successes and failures of space exploration.

Reflection

Phase 1

Our team did a good job of learning the required skills to develop the website. For this phase, we had to learn how to use React to develop the frontend, how to deploy on AWS and how to utilize API calls to pull data. We were able to successfully separate the tasks so that each person did not have to learn every single thing. This gave them the opportunity to spend most of their time on a specific portion and become more well-versed in the tool, software, language, etc. Another thing we did well was finding the data for our project. We were able to find various datasets with very pertinent information for our users. However, there are some things we can improve on. One thing is becoming set on what data we want to display to the user. At the moment, we are displaying the data from our datasets, but not all of it is necessary so we need to go through the data to see what we can ignore. This way, the website is more

digestible for our users. Additionally, we need to become more organized in how we approach the production of our features. There have been instances where we try to implement several features at the same time and this impacts our design since we have to account for several things at the same time. If we did things in a more linear fashion, our design process would probably not be as complicated. Overall, we learned to start early and draw out what we want to do because visualizing the projects makes implementation easier.

Phase 2

For phase two our team had to learn several different skills to implement the necessary requirements. These skills were testing, backend/API development, and database management. We split up the responsibilities between the group members which we found to be very effective. This is because each person had more time go into depth with a certain topic rather than trying to learn every single thing. For example, Shawn and Lucas were able to test a lot of the code very quickly because that was one of their main focuses. Meanwhile, Jack was able to set up the frontend aspects of a feature while I simultaneously developed the backend API for said feature. However, things were rough at the beginning of the phase for us because of the recent pandemic. For instance, it became hard to communicate and collaborate due to everyone moving away, but we were able to adapt to the circumstances by having meetings on Zoom. Overall, it will be challenging these next two phases with everyone having to work remotely, but it will be a great learning experience for when we go into the real world.

Phase 3

For phase three the team did not have to take on too many new technologies or skills to fulfill the requirements. Most of the time was spent making the system more robust. For example, we migrated our API data to the database rather than pulling directly from the API on our frontend. This caused our information to show up quicker, especially for our launches. Before, it took close to thirty seconds to load the launch information and now it takes about five or six seconds. On the front end, we focused on changing the layout of our model pages to look more appealing. In terms of communication, we did a much better job this phase meeting on Zoom and giving updates on Slack.