# Software Design Document for VIPER Rocks!

Version 1.0.0 approved

Prepared by Kevin Andrade, Diana Arteaga-Andrade, Santiago Bautista, Michael Gibson, Cristian Gomez, Nida Sheikh, Zainab Sulaiman, Diane Audin Tabilas, Angy Xajil, Tammy Xaypraseuth, and Jerome Pineda

Sponsored by NASA JPL

Created on 10/23/23

# Table of Contents

# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| Entire Team | Dec. 8, 2023 | SDD Draft | 1.0. |
| | | | |
| | | | |
| | | | |

<Add rows as necessary when the document is revised. This document should be consistently updated and maintained throughout your project. If ANY requirements are changed, added, removed, etc., immediately revise your document.>

# 1. Introduction

## 1.1 Purpose

Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.

Product: VIPER Rocks! Web Application
Revision/Release Number: v0.0.1 (as of December 08, 2023)
This document covers the core functionalities of the VIPER Rocks! citizen science application, including the scouting task, sizing task, classification task, account creation, badges, user interface, and database. This document will go over how to use the application as well as the

## 1.2 Document Conventions

The typographical conventions of this document are basic. The font is Times New Roman. Each higher-level requirement is to be inherited by the detailed requirement.

## 1.3 Intended Audience and Reading Suggestions

This document is intended for a diverse group of readers, each with their specific needs and interests. Here's a breakdown of the different types of readers and their primary concerns:

- Software Developers
    - For the people who want to understand and update the technical specifications and architecture of the software.
- Project Managers
    - For them to understand how the software components interact to fulfill each requirement
- Users
    - Learn to use the software effectively and understand its capabilities and limitations
- Testers
    - Understand the software's requirements and design to plan and execute effective test cases.

The sequence presented in this document is the intended sequence to read the document.

## 1.4 System Overview

VIPER Rocks! is a web-based application for JPL that gathers data given by citizen scientist users. Citizen scientists are tasked to look at pictures given by VIPER, NASA's rover being sent to the South Pole of the Moon, and to tag and complete three tasks given to them.
The three tasks are the following:

- Scouting:

    - A citizen scientist will need to tag the rocks within a given image. The image is then split up evenly for users to do the other tasks
- Sizing:
    - A citizen scientist will be given tools to measure the size of a particular rock.
- Classification:
    - A citizen scientist will be asked to identify a particular rock as either of four categories: angular, sub-angular, rounded, or sub-rounded.

Whenever each task is accomplished it will be recorded in the database.

# 2. Design Considerations

## 2.1 Assumptions and Dependencies

Assumptions:

- Users will have basic computer literacy and familiarity with using web browsers and mobile applications.
- Users will have reliable internet access to participate in the application.
- NASA will provide timely and accurate data and images from the VIPER mission.
- The application will function properly on a wide range of devices with varying specifications.
- The application will be compatible with the latest versions of popular operating systems, including Windows, macOS, iOS, and Android.

Dependencies:

Software Dependencies:

- React 18.2.0
- MySQL 8.0
- MongoDB 7.0
- Python 3.12.0
- Google Chrome
- Safari
- Firefox

Hardware Dependencies:

- Latest versions of popular browsers like Chrome, Firefox, Safari
- Mobile devices or personal computers with sufficient processing power and memory
- Reliable internet access for accessing the application and data

Possible Changes in Functionality:

- Based on user feedback and evolving needs, the application may be expanded with new features and functionalities.
- The application may be adapted to support future lunar exploration missions beyond VIPER.
- Updates to operating systems and browsers may require adjustments to the application to maintain compatibility.

## 2.2 General Constraints

The major constraint of this software will be accessibility. Accessibility will be the number one priority while designing any of the interactions that will happen on the UI/UX.

Another constraint will be designing for simplicity on the UI/UX. This ultimately means that anything developed should be only a one-click input.

Hardware and Software Environment:

- Resource limitations: Users may participate using diverse devices with varying processing power and memory capacity. This necessitates a lightweight and efficient application design to ensure optimal performance across various hardware configurations.
- Software compatibility: The application must be compatible with a range of operating systems and browsers.

End-user Environment:

- Technical proficiency: The application should be user-friendly and intuitive, catering to users with diverse technical backgrounds. Extensive user testing and feedback are crucial to ensure accessibility and usability.

Availability and Volatility of Resources:

-

## 2.3 Goals and Guidelines

The Viper Rocks! website will follow UI/UX guidelines set by NASA. Since this website will be part of NASA, their aesthetics will be followed.

The website will also be following another NASA guideline which pertains to accessibility.

The design of the system will follow the KISS principle, Keep it Simple Stupid.

## 2.4 Development Methods

The methodology that was implemented for the design of this website is the Water Fall Development method. During this first semester, all requirements have been captured and will be executed during the second semester.

However, during the second semester, the plan will be Agile Development. The reasoning behind this is that the design is still ambiguous during this time. With Agile Development we shall be able to design and test our UI/UX design quicker and better during each sprint. We shall be able to get feedback in a timely matter and get the needed data to improve our design

# 3. Architectural Strategies

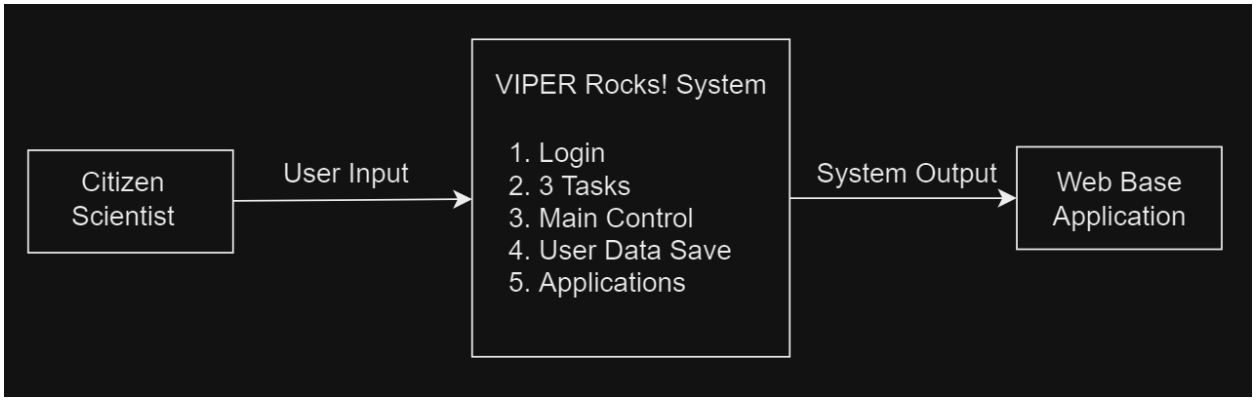
The design of the website will be split up between front-end development and back-end development.

- Front-end will handle UI/UX.
- Back-end will handle the database and API calls.

The front end will be coded in with the javascript library, React. React was chosen due to its component-based nature allowing to split the UI design faster and deploy quickly. Another reason is because of Reacts many libraries will reduce the amount of work done when implementing the UI design

The back end will be coded in SQL. SQL was chosen due to familiarity with the language itself.

# 4. System Architecture



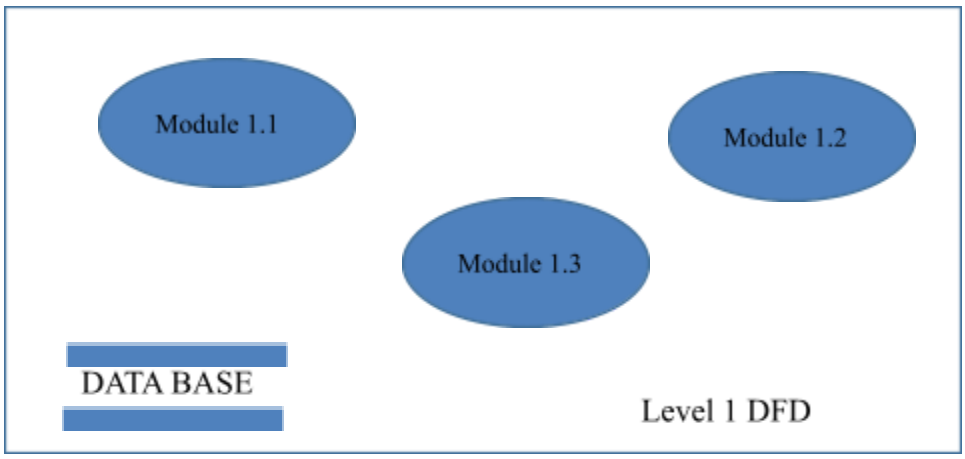

At the top-most level, describe the major responsibilities that the software must undertake and the various roles that the system (or portions of the system) must play. Describe how the system was broken down into its modules/components/subsystems (identifying each top-level modules/component/subsystem and the roles/responsibilities assigned to it).

Each subsection (i.e. “4.1.3 The ABC Module”) of this section will refer to or contain a detailed description of a system software component.

**VIPER Rocks! System** encourages citizen scientists to assist further research on surface rocks. The system is seen as a tool; a bank of information that could help determine future NASA exploration missions with the help of citizen scientists.

**VIPER Rocks! System** is a website driven project that provides significant functions such as the ones listed below:

- Controls and regulates access to the website through the log-in/security function
- Provides networked access to a variety of UI pages
- Manages the transfer of data and communication within a network
- Provides functionality for a range of interface-specific tools
- Supports an assortments of powerful essential utilities




Level 1 Data Flow Diagrams (DFD) and Control Flow Diagrams (CFD) should probably go here.

Describe how the higher-level components collaborate with each other in order to achieve the required results. Don't forget to provide some sort of rationale for choosing this particular decomposition of the system (perhaps discussing other proposed decompositions and why they were rejected). Feel free to make use of design patterns, either in describing parts of the architecture (in pattern format), or for referring to elements of the architecture that employ them. Diagrams that describe a particular component or subsystem in detail should be included within the particular subsection that describes that component or subsystem.

# 5. Policies and Tactics

Describe any design policies and/or tactics that do not have sweeping architectural implications (meaning they would not significantly affect the overall organization of the system and its high-level structures), but which nonetheless affect the details of the interface and/or implementation of various aspects of the system. Make sure that when describing a design decision that you also discuss any other significant alternatives that were considered, and your reasons for rejecting them (as well as your reasons for accepting the alternative you finally chose). Such decisions might concern (but are not limited to) things like the following (Must include 5.1, 5.2, and 5.3. The rest of these categories or custom ones can be added as needed.):

## 5.1 Choice of which specific products used

(IDE, compiler, interpreter, database, library, etc. ...)

## 5.2 Plans for ensuring requirements traceability

**…Describe…**

## 5.3 Plans for testing the software

…Describe…

5.# Engineering trade-offs

…Describe…

5.# Coding guidelines and conventions

…Describe…

5.# The protocol of one or more subsystems, modules, or subroutines

…Describe…

5.# The choice of a particular algorithm or programming idiom (or design pattern) to implement portions of the system's functionality

…Describe…

5.# Plans for maintaining the software

…Describe…

5.# Interfaces for end-users, software, hardware, and communications

…Describe…

5.# Hierarchical organization of the source code into its physical components (files and directories).

…Describe…

5.# How to build and/or generate the system's deliverables (how to compile, link, load, etc.)

…Describe…

5.# Describe tactics such as abstracting out a generic DatabaseInterface class, so that changing the database from MySQL to Oracle or PostGreSQL is simply a matter of rewriting the DatabaseInterface class.

For this particular section, it may become difficult to decide whether a particular policy or set of tactics should be discussed in this section, or in the System Architecture section, or in the Detailed System Design section for the appropriate component. You will have to use your own "best" judgement to decide this. There will usually be some global policies and tactics that should be discussed here, but decisions about interfaces, algorithms, and/or data structures might be more appropriately discussed in the same (sub) section as its corresponding software component in one of these other sections.

# 6. Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

This is where Level 2 (or lower) DFD's will go. If there are any additional detailed component diagrams, models, user flow diagrams or flowcharts they may be included here.

## 6.1 Scouting Task (Module)

### 6.1.1 Responsibilities

The primary responsibility of the scouting task is to retrieve information about where the lunar rocks are gathered. These precise coordinate points are going to help both NASA and citizen scientists in being able to have exact locations of where the lunar rocks are situated on the moon's surface.

This component is going to provide a user-friendly interface for individuals to input their diverse observations regarding the criteria for moon rock scouting. The scouting task will include advanced search and filtering options allowing users to specify the geological features for a refined exploration. There will be images displayed, descriptions and relevant details about the scouted lunar rock to produce accurate results for the data collection.

### 6.x.2 Constraints

Some of the constraints regarding the scouting tasks are acquiring precise coordinates, including sufficient storage, providing a standardized and consistent format for input data, and interaction rules. These constraints will enable the website to operate within reasonable boundaries yet maintain a user-friendly experience.

### 6.x.3 Composition

The Database Query and Retrieval is an essential subcomponent because it will be responsible for obtaining the user's findings in real-time to be sent off to the other tasks such as sizing and classification.

### 6.x.4 Uses/Interactions

The user interface serves as the starting point to navigate the moon images and will direct users to the scouting area where they will input their observations. Other components that

are used in this entity are scouting algorithms and map integrations. The scouting task requires an interactive map/picture to get an idea of where the lunar rocks are located when scouted.

### 6.x.5 Resources

Some resources that can be affected during the launch of the website include the network, server, and database resources. First, having network issues can impact real-time interaction and responsiveness when attempting to complete the scouting task. Second, the server load could impact the speed and responsiveness of the scouting operations. Third, the database resources affect the quality, accuracy, and reliability if there aren't enough resources. Lastly, one must take into account race conditions and deadlock situations because they could arise when multiple users are interacting with the system.

### 6.x.6 Interface/Exports

- Scout Lunar Rocks:
  - Definition: Initiates the lunar rock scouting process based on the provided criteria.
    - Responsibilities:
- Get Scouting Results:
  - Definition: Receives the current set of scouting results
    - Responsibilities:
- Export Scouting Data:
  - Definition: Allow users to export data in a specified format
    - Responsibilities:
- Handle User Feedback:
  - Definition: Manage and process user feedback
    - Responsibilities:

## 6.2 Sizing Task (Module)

### 6.x.1 Responsibilities

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements specification.

### 6.x.2 Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, post conditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

### 6.x.3 Composition

A description of the use and meaning of the subcomponents that are a part of this component.

### 6.x.4 Uses/Interactions

A description of this components collaborations with other components. What other components is this entity used by? What other components does this entity use (this would include any side-effects this entity might have on other parts of the system)? This concerns the method of interaction as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated subclasses, superclass's, and metaclasses.

### 6.x.5 Resources

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

### 6.x.6 Interface/Exports

The set of services (classes, resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

## 6.3 Classification Task (Module)

### 6.x.1 Responsibilities

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide

to its clients? For some components, this may need to refer back to the requirements specification.

### 6.x.2 Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, post conditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

### 6.x.3 Composition

A description of the use and meaning of the subcomponents that are a part of this component.

### 6.x.4 Uses/Interactions

A description of this components collaborations with other components. What other components is this entity used by? What other components does this entity use (this would include any side-effects this entity might have on other parts of the system)? This concerns the method of interaction as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated subclasses, superclass's, and metaclasses.

### 6.x.5 Resources

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

### 6.x.6 Interface/Exports

The set of services (classes, resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

## 6.4 Account Creation (Module)

### 6.x.1 Responsibilities

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements specification.

### 6.x.2 Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, post conditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

### 6.x.3 Composition

A description of the use and meaning of the subcomponents that are a part of this component.

### 6.x.4 Uses/Interactions

A description of this components collaborations with other components. What other components is this entity used by? What other components does this entity use (this would include any side-effects this entity might have on other parts of the system)? This concerns the method of interaction as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated subclasses, superclass's, and metaclasses.

### 6.x.5 Resources

A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

### 6.x.6 Interface/Exports

The set of services (classes, resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. For each service element described, include (or provide a reference) in its discussion a description of its important software component

attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

Much of the information that appears in this section is not necessarily expected to be kept separate from the source code. In fact, much of the information can be gleaned from the source itself (especially if it is adequately commented). This section should not copy or reproduce information that can be easily obtained from reading the source code (this would be an unwanted and unnecessary duplication of effort and would be very difficult to keep up-to-date). It is recommended that most of this information be contained in the source (with appropriate comments for each component, subsystem, module, and subroutine). Hence, it is expected that this section will largely consist of references to or excerpts of annotated diagrams and source code.

# 7. Detailed Lower level Component Design

TBD

Other lower-level Classes, components, subcomponents, and assorted support files are to be described here. You should cover the reason that each class exists (i.e. its role in its package; for complex cases, refer to a detailed component view.) Use numbered subsections below (i.e. “7.1.3 The ABC Package”.) Note that there isn't necessarily a one-to-one correspondence between packages and components.

## 7.x Name of Class or File

### 7.x.1 Classification

The kind of component, such as a subsystem, class, package, function, file, etc.

### 7.x.2 Processing Narrative (PSPEC)

A process specification (PSPEC) can be used to specify the processing details

### 7.x.3 Interface Description

### 7.x.4 Processing Detail

#### 7.x.4.1 Design Class Hierarchy

Class inheritance: parent or child classes.

#### 7.x.4.2 Restrictions/Limitations

#### 7.x.4.3 Performance Issues

#### 7.x.4.4 Design Constraints

### 7.x.4.5 Processing Detail For Each Operation

# 8. Database Design

Include details about any databases used by the software. Include tables and descriptions.

# 9. User Interface

The user interface is the application, from the point of view of the users. Do your classes and their interactions (the logical and process views) impose restrictions on the user interface? Would removing some of these restrictions improve the user interface? Use some form of user interface flow model to provide an overview of the UI steps and flows. Don't go into too much refinement. You should include screen shots or wireframe layouts of significant pages or dialog elements. Make sure to indicate which of the system level modules or components that each of these user interface elements is interacting with.

## 9.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback Information that will be displayed for the user. This is an overview of the UI and its use. The user manual will contain extensive detail about the actual use of the software.

From the user's perspective, VIPER Rocks! presents a seamless and user-friendly interface that facilitates an engaging and educational experience. The initial step involves creating an account where the users above 12 provide a strong password, stored securely with salting and hashing. For younger users, parental involvement is required, emphasizing data collection for scientific purposes while ensuring privacy.

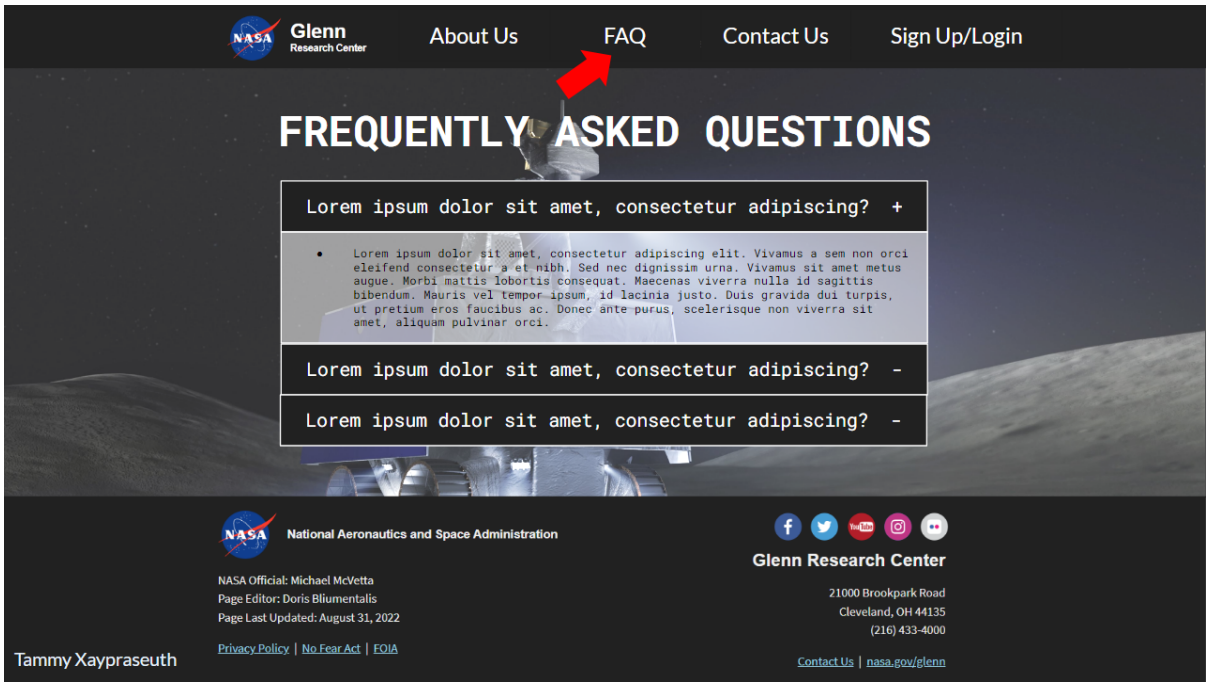
The home page welcomes users with an optional tutorial and showcases VIPER's mission, purpose, and user feedback. Notifications from the science team, banner announcements, and contact information for bug reports or queries from JPL add transparency. Integration with OAuth, Facebook, Google login and an institution feature for schools to enhance the community aspect.

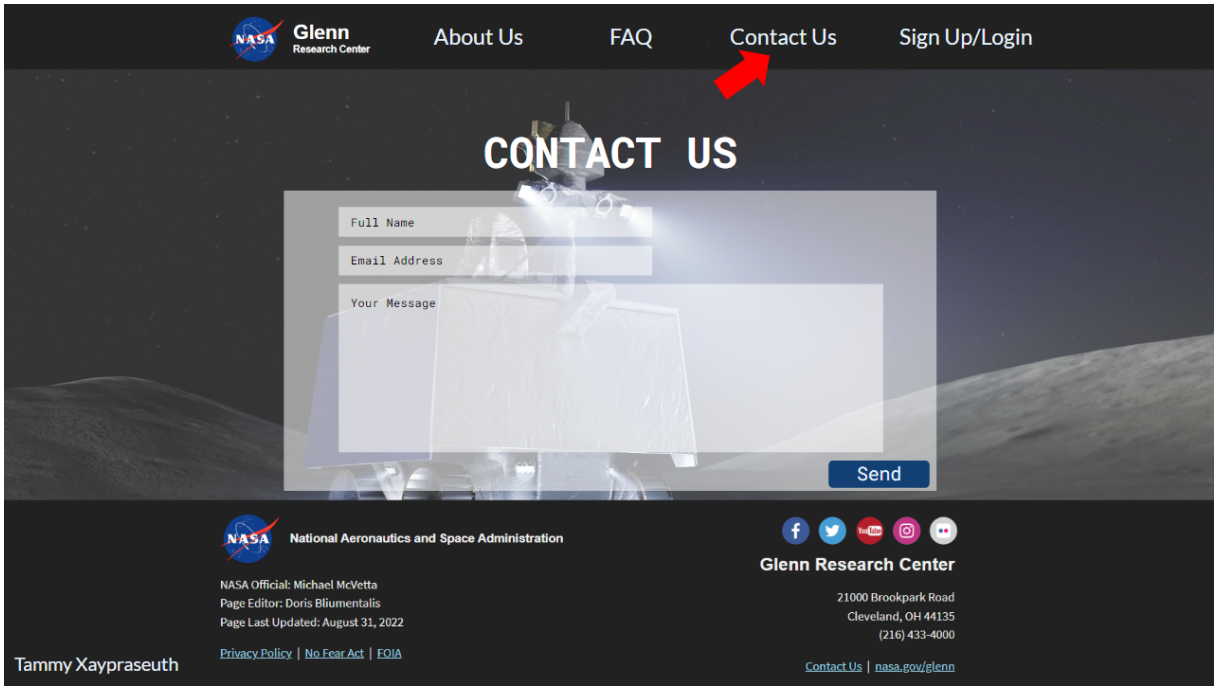
The scouting page introduces distribution of rocks within an image and a sizing task with intuitive desktop and mobile tools that enrich the scouting experience that lead to a pop-up menu for the classification task. The general design incorporates a side daw navigation bar for tools, offering a unified experience across mobile and desktop platforms.

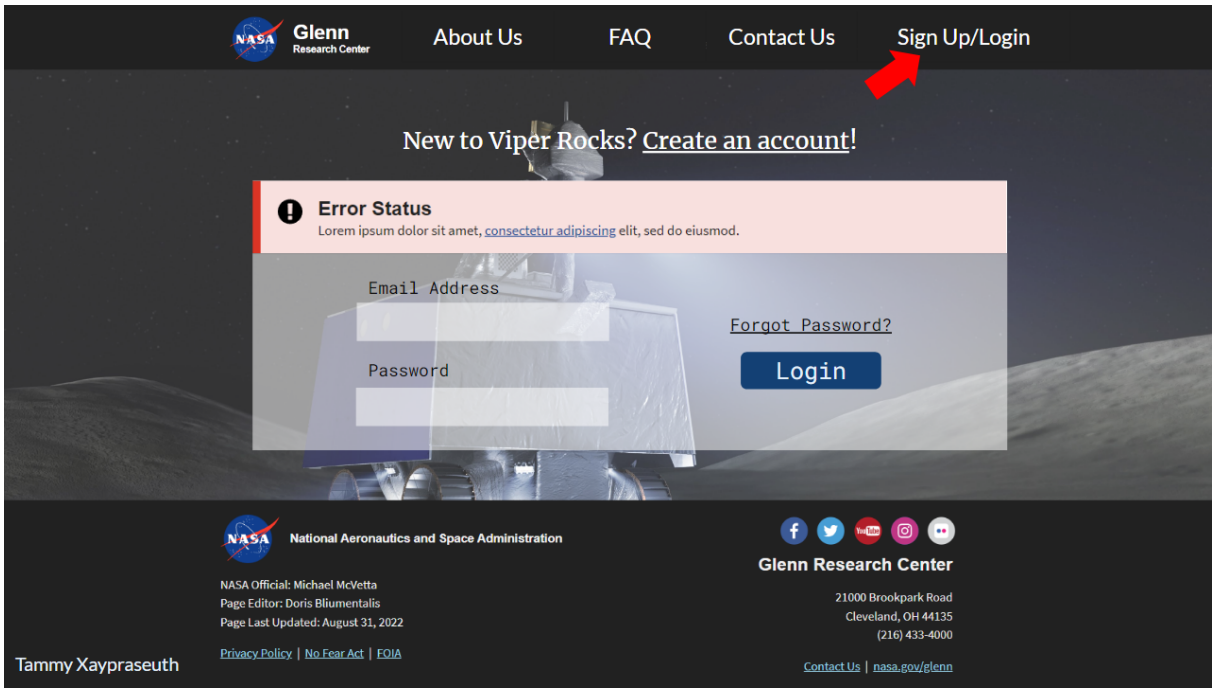
In summary, VIPER Rocks! aims to empower users to contribute meaningfully to scientific endeavors, providing a well-designed, educational, and rewarding platform while ensuring data security and accuracy. The detailed user manual supplements this overview, offering comprehensive guidance on system functionality.

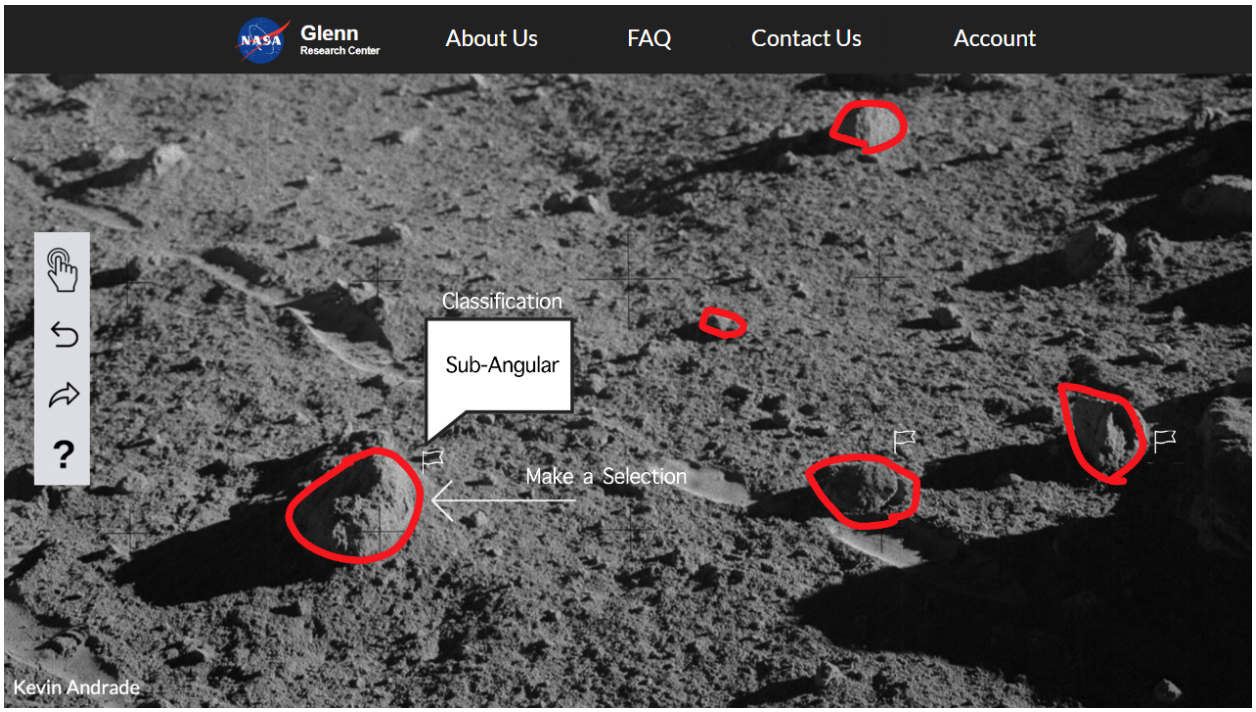
## 9.2 Screen Frameworks or Images

These can be mockups or actual screenshots of the various UI screens and popups.


Glenn
Research Center
About Us
FAQ
Contact Us
Sign Up/Login
CONTACT US
Full Name
Email Address
Your Message
Send
National Aeronautics and Space Administration
NASA Official: Michael McVetta
Page Editor: Doris Bliumentalis
Page Last Updated: August 31, 2022
Privacy Policy | No Fear Act | FOIA
Glenn Research Center
21000 Brookpark Road
Cleveland, OH 44135
(216) 433-4000
Contact Us | nasa.gov/glenn
Tammy Xaypraseuth


Glenn
Research Center
About Us
FAQ
Contact Us
Sign Up/Login
New to Viper Rocks? Create an account!
Error Status
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod.
Email Address
Password
Forgot Password?
Login
National Aeronautics and Space Administration
NASA Official: Michael McVetta
Page Editor: Doris Bliumentalis
Page Last Updated: August 31, 2022
Privacy Policy | No Fear Act | FOIA
Glenn Research Center
21000 Brookpark Road
Cleveland, OH 44135
(216) 433-4000
Contact Us | nasa.gov/glenn
Tammy Xaypraseuth




## 9.3 User Interface Flow Model

A discussion of screen objects and actions associated with those objects. This should include a flow diagram of the navigation between different pages.

# 10. Requirements Validation and Verification

Create a table that lists each of the requirements that were specified in the SRS document for this software.
For each entry in the table list which of the Component Modules and if appropriate which UI elements and/or low level components satisfies that requirement.
For each entry describe the method for testing that the requirement has been met.

# 11. Glossary

An ordered list of defined terms and concepts used throughout the document. Provide definitions for any relevant terms, acronyms, and abbreviations that are necessary to understand the SDD document. This information may be listed here or in a completely separate document. If the information is not directly listed in this section provide a note that specifies where the information can be found.

# 12. References

<List any other documents or Web addresses to which this SDD refers. These may include other SDD or SRS documents, user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

Brad Appleton <brad@bradapp.net> http://www.bradapp.net

https://www.cs.purdue.edu/homes/cs307/ExampleDocs/DesignTemplate_Fall08.doc