
# JSON Validator

Submitted by **TEAM 23**

| | |
|---|---|
| **Abhay Kaushik** | 2022201054 |
| **Ayush Lakshkar** | 2022201051 |
| **Advait Shrivastava** | 2022201069 |
| **Prem Sai Kumar** | 2022201036 |

Under the Supervision of

**Mr. Sai Anirudh Karre**

Research Scholar

SERC Lab

## Description

- Software practitioners use a variety of Requirement engineering approaches to produce a well-defined product. These methods impact the software product's ultimate traits. VR has become an essential technology for the future. Nevertheless, very few tools and methods practiced for requirement engineering are not capable enough of addressing all aspects of VR product development. Here we attempt to develop a requirement specification tool for VR system development.

## Requirements

- The VReqst tool is aimed to reduce the developer creation time by taking input from the customer and validating it against the constraints mentioned by the admin of the organization, to ensure the the requirements of the customer are valid. There is an integrated rules editor framework that allows the user to write code faster by giving code snippets.
- To cater to the above requirements, the automated validation method are developed that validates the requirements at five stages along with the rule editor framework to code logic that stores the final results in the database for later viewing and downloads.

# Technology Stack

## Storage:

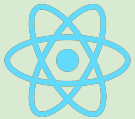
- MongoDB Atlas

## Back-end:

- Node
- Express


## Front-end:

- React
- JS



## Login Management

- Created interface for login, registration and user authentication.

- Created frontend validation to validate field entries.

- Created all required endpoints in React.

- An admin is predefined for an account for the organization.

- User is created by entering the details.

- After login, the list of projects associted with the user will be displayed. User can also create new project.

- Dashboard at login shows the list of projects

## Workflow

- Each project will have an edit or view button. The edit button is available as long we have JSONS to validate.

- View enables user to view the edited and saved json till that point.

- After uploading the 3 JSONS: Scene, Asset and Action, the tool will be redirected to the portal for custom rule editor framework. Here the user can use the code snippets to write the code and check if the the syntax is valid or not.

- Rule editor generates the base 64 encoded strings and will be uplaoded to mongo db collection.

- After generating base64 encoded strings, tool redirects the user to validate other JSONs i.e Custom JSON and Scene Timeline. Base64 encoded strings will be used to fill Custom JSON.

## List of MongoDB Collections

**Step:** keep track of the current step upto which the project is validated and the last step which was validated.

**Session:** Store the session of the logged in user at what time he/she logged in and out.

**Project Model:** Stores projects of all users with creation time,status, last updated time.

**User JSON Model:** Stores the valid user JSONs that has been uploaded by the user. It also stores the project ID and step number of the step of the project number.

**Validator Model:** Stores all the valid JSON uploaded by the admin.

**Integrate Model:** Stores the list of BASE64 encoded strings sent from the separate Rule Editor Framework project used further in validating user input custom JSON in our project.

**PersonModel:** Stores the details of the person.It also stores the role of the person.

**LoginModel:** Stores the login details of the user.It also stores the role of the user.

**ProjectModel:** **S**tores details like owner, project id, creation time, status of the project.

## Grammar Parsing and Scene Validation

- Validated scene definition script for given input JSON.

- Checked for optional fields and typeof input.

- The validation code takes the input JSON and the validator JSON as input and then checks if the input JSON has a JSON that is valid according to the validator JSON.

- In case the input JSON has nested JSON objects or arrays, we send a recursive function to check if they values have the same datatype as the one mentioned inside validator JSON.

# MongoDB Collections Screenshots

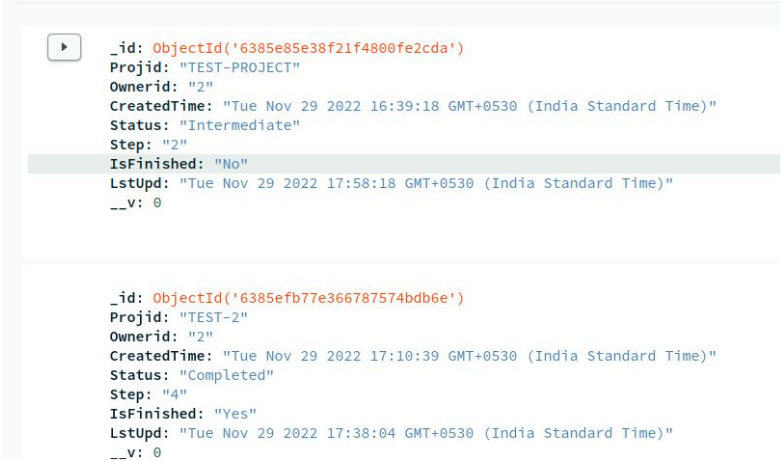

```
_id: ObjectId('6385ea6a38f21f4800fe2cfe')
input_json_content: "{
                        "_scenename":"Bowlling Alley",
                        "_sid":"s45189",
                        "_slabel":"…"
pid: "2"
step_no: "0"
proj_id: "TEST-PROJECT"
__v: 0
```

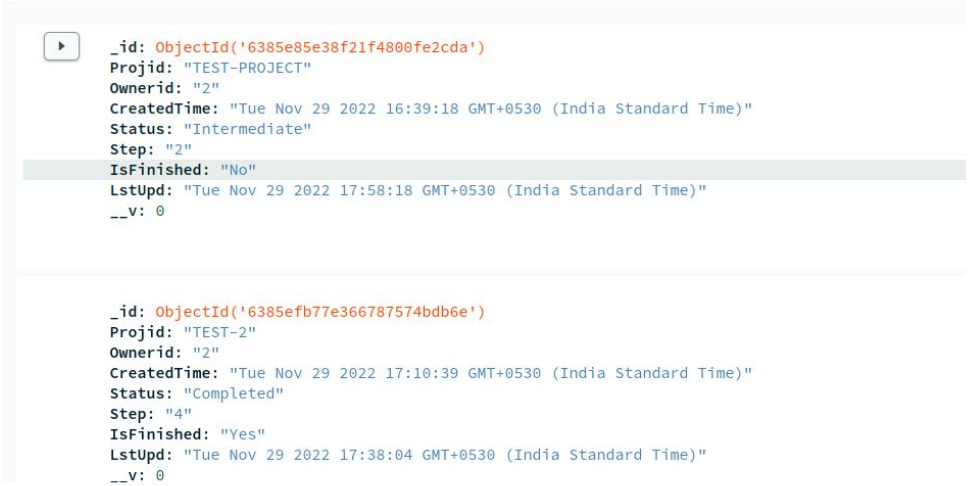
```
_id: ObjectId('6385eff87e366787574bdb75')
input_json_content: "{
                        "_scenename":"Bowlling Alley",
                        "_sid":"s45189",
                        "_slabel":"…"
pid: "2"
step_no: "0"
proj_id: "TEST-2"
  v: 0
```

```
_id: ObjectId('6369502e6f913e3e6f9bc7f2')
pid: "200"
firstname: "admin"
lastname: "admin"
email: "admin@admin.com"
organization: "iiith"
role: "1"
__v: 0
```

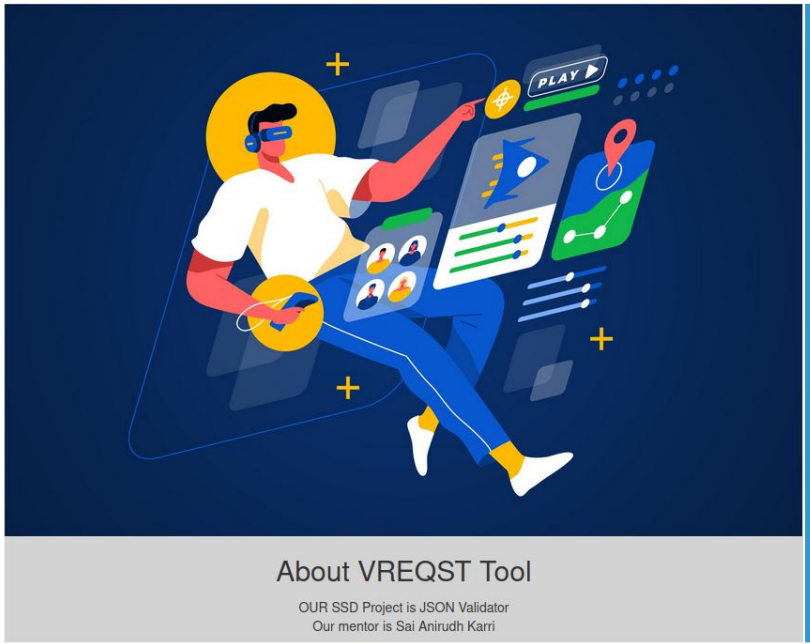
```
_id: ObjectId('6385e7f038f21f4800fe2cd1')
pid: "2"
firstname: "Advait"
lastname: "Shrivastava"
email: "advait_shrivastava@students.iiit.ac.in"
organization: "IIITH"
role: "0"
__v: 0
```

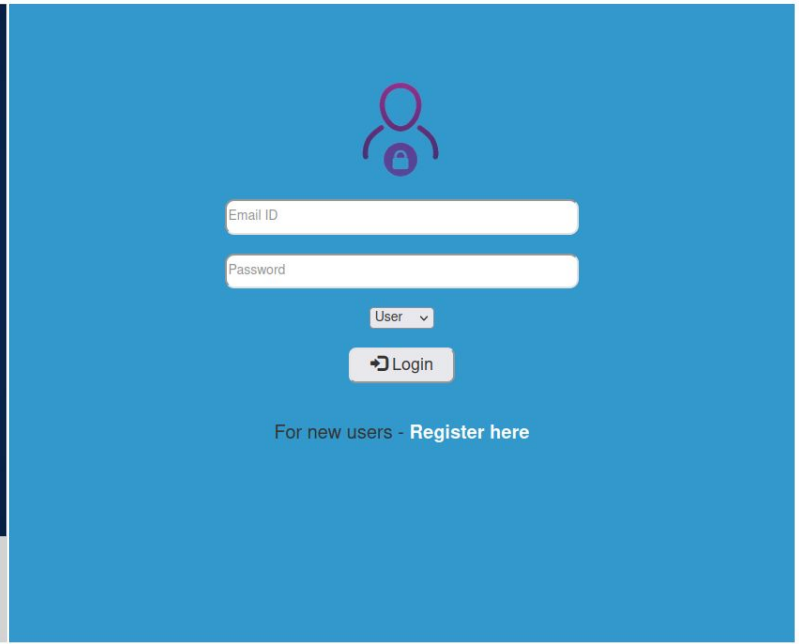
# MongoDB Collections Screenshots



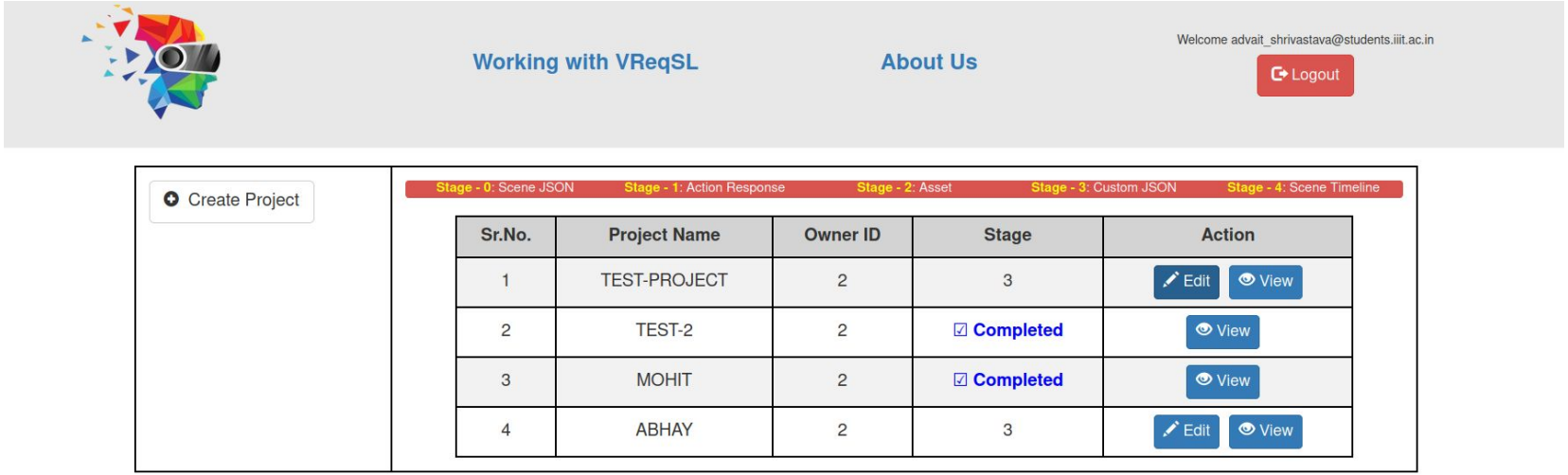
# WebApp Screenshots




Login Page

# WebApp Screenshots

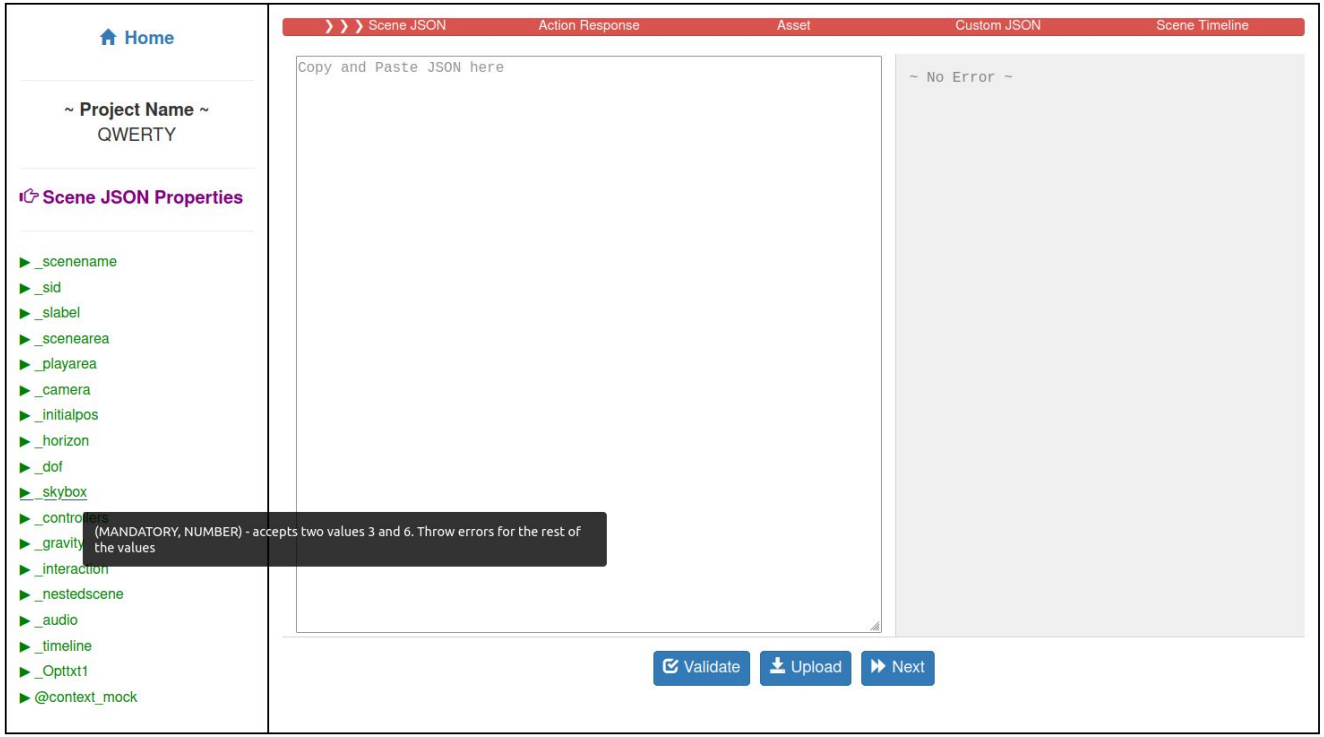



Project Dashboard

# WebApp Screenshots

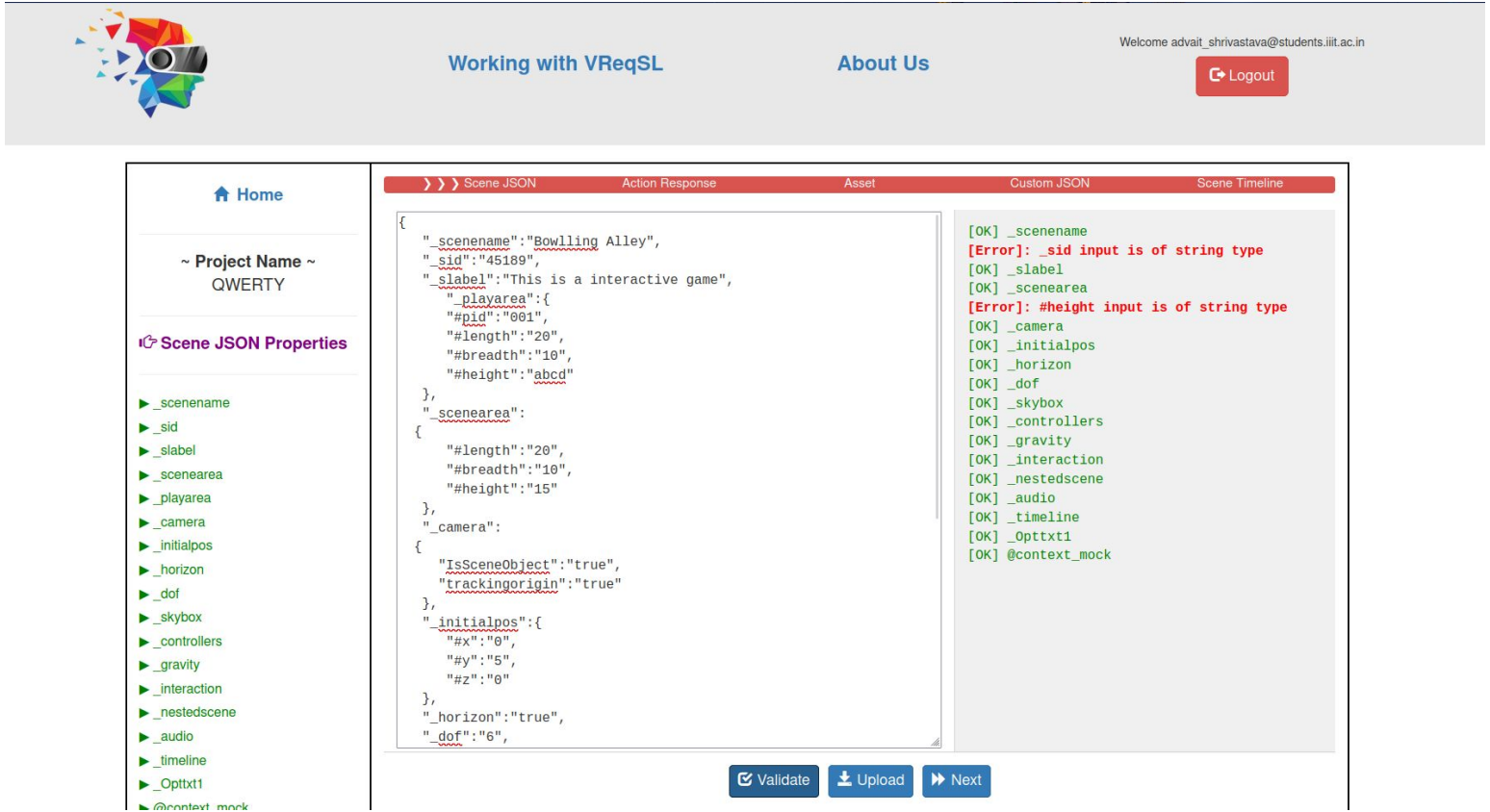

Create Project

# WebApp Screenshots



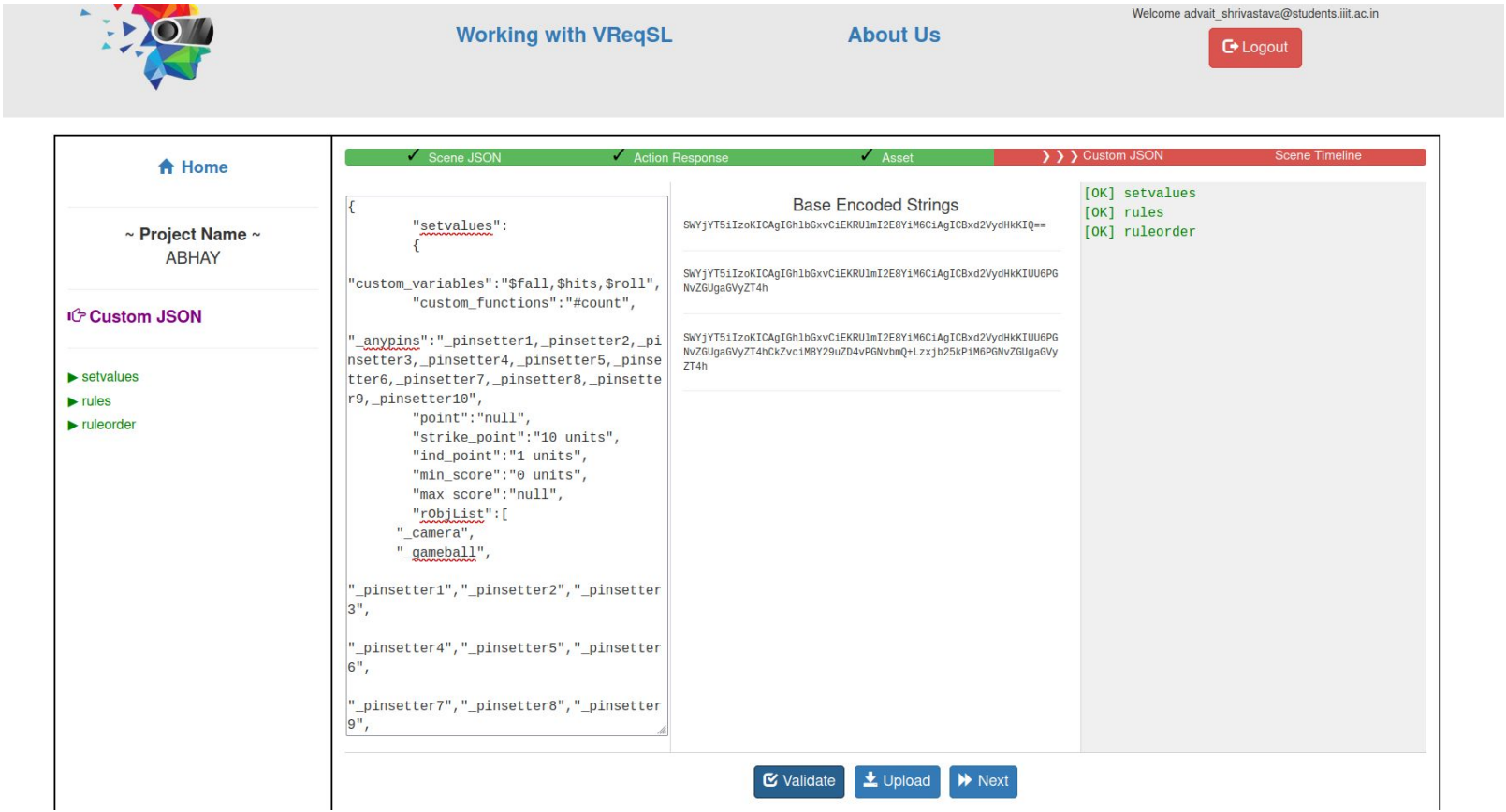

JSON Validation Page

# WebApp Screenshots



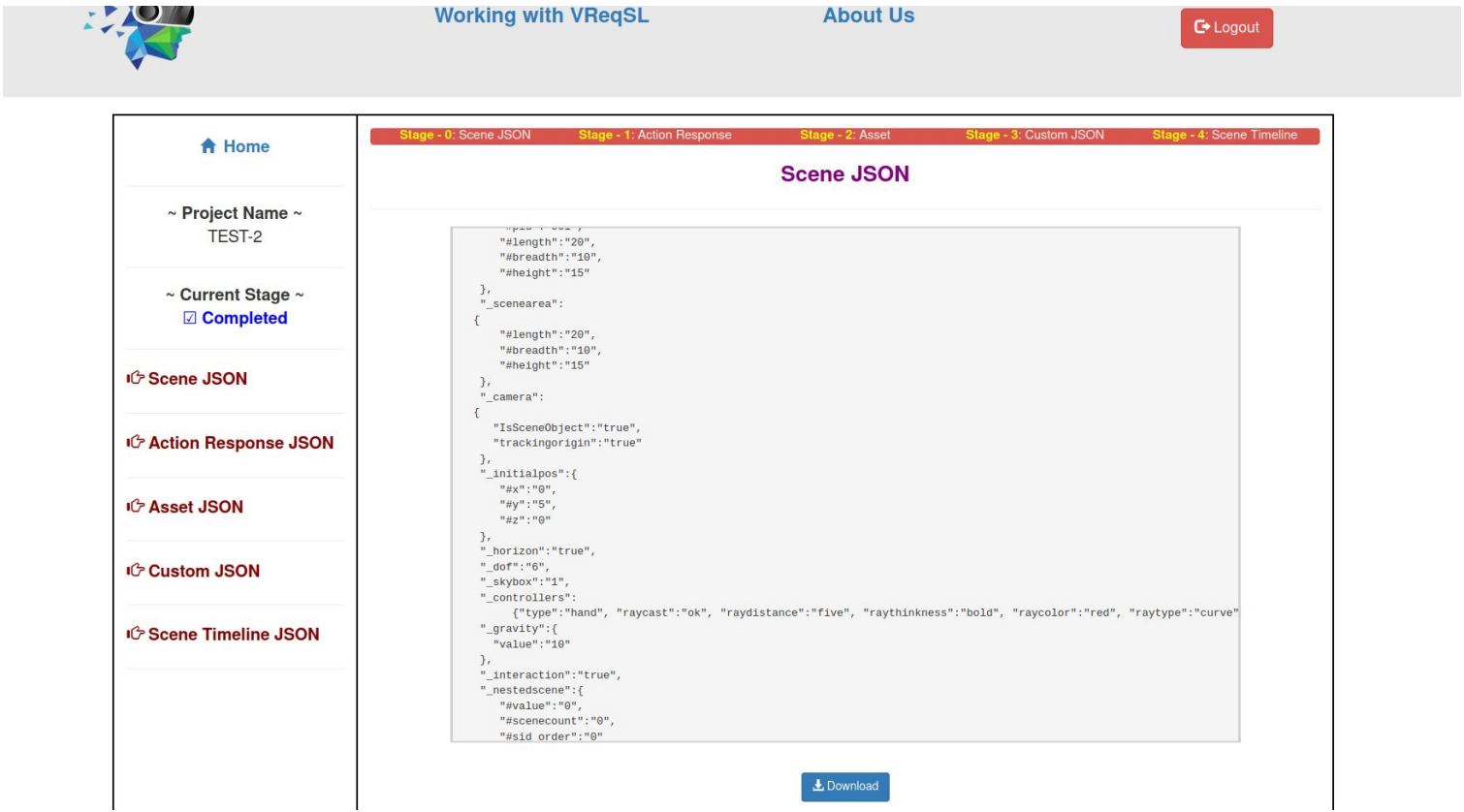

JSON Validation Page

# WebApp Screenshots




Base64 Encoded Strings Page

# WebApp Screenshots




View Project Page

# JS Code Snippet

```
const validate = (jsonObject, validator) => {

    // get the list of keys of the sceneValidator object
    let validator_keys = Object.keys(validator);
    let flag = true; // assume valid entries in jsonObject
    for (let k of validator_keys) {
        if(jsonObject[k] == undefined) {
            // the field is not there, so we check if it is mandatory
            if(validator[k].req == "mandatory") {
                // the field is mandatory, so we throw error
                console.log("Error: Input JSON doesn't have field " + k);
            }
            // else the field is optional, so we continue
        }
        else {
            // we check if the properties are valid
            if(typeof jsonObject[k] == "object") {
                // console.log("Object : " + jsonObject[k])
                if(Array.isArray(jsonObject[k] == true)) {
                    // it is an Array Object
                    flag = check_type_array(jsonObject[k], validator[k].typeof)
                }
                else {
                    // it is a JSON Object
                    flag = check_type(jsonObject[k], validator[k].typeof)
                }

                if(flag == true) {
                    console.log("[OK] " + k)
                }
            }
            else {
                if(extract(jsonObject[k]) != validator[k].typeof) {
                    // property type is invalid
                    console.log("Error: " + k + " input is of " + typeof jsonObject
                }
                else {
                    // property type is valid
                    console.log("[OK] " + k)
```

```
[OK] _scenename
Error: _sid input is of string type
[OK] _slabel
Error: Input JSON doesn't have field _scenearea
Error: #pid input is of string type
Error: trackingorigin input is of string type
[OK] _initialpos
[OK] _horizon
[OK] _dof
Error: _skybox input is of string type
Error: raycast input is of string type
Error: raydistance input is of string type
Error: raythinkness input is of string type
[OK] _gravity
[OK] _interaction
Error: #value input is of string type
Error: #scenecount input is of string type
Error: #sid_order input is of string type
[OK] _audio
[OK] _timeline
[OK] _Opttxt1
[OK] @context_mock
Error: _viewport is not part of validator JSON
Error: _clippingplane is not part of validator JSON
object
```

# Thank You