

Team member: Toh Zeng Hau (1843088)

Kwan Chi Hong (1843062)

Lecturer: Mr Ji Peng Xiang

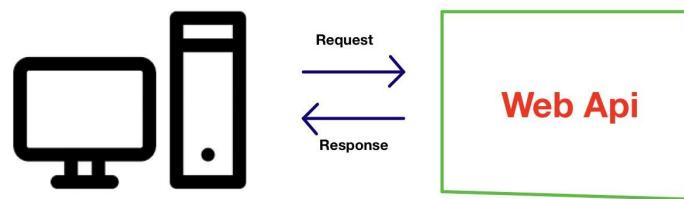
Title: CSC Assignment 1

Table of Contents

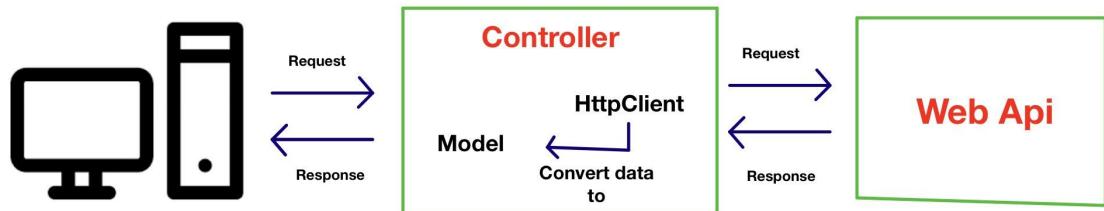
Task 1 :	3
Figure 1.1 - Difference between consuming web api with Jquery Ajax and C#	4
Figure 1.2 - Consuming web api with Ajax	5
Figure 1.3 - Consuming web api with C#	5
Task 2 :	6
API Documentation:	6
Figure 2.1 - Get all product API	6
Figure 2.2 - Get product with ID API	6
Figure 2.4 - Add new product API	7
Figure 2.5 - Update a product API	8
Figure 2.6 - Delete product API	8
Postman Results and Screenshots:	9
Figure 2.7 - Get all the product from application	9
Figure 2.8 - Get Products by ID	9
Figure 2.9 - Get Products by Category	10
Figure 2.10 - Create a product	10
Figure 2.11.2 - Result of updated product has been added to the product list successfully	11
Figure 2.12 - Delete product from the list	12
Figure 2.13 - Overposting	12
Figure 2.14 - Underposting	13
Task 3:	14
API Documentation:	14
Figure 3.1 - Get Values API	14
Figure 3.2 - Logout Account API	14
Figure 3.3 - Register Account API	15
Postman Testing and Screenshot :	15
Figure 3.4.1 - Register an user account with required params attached	15
Figure 3.4.2 - The recaptcha verify api is called with GoogleCaptchaToken and secret key attached then a score will be returned to state whether the action is valid or not.	16
Figure 3.5 - Login with the proper credentials	16
Figure 3.6 - Retrieve the result message with Bearer token attached in the request header	17

Figure 3.7 - Logout from the application	17
Figure 3.8 - Sequence Diagram	18
Task 4:	18
API Documentation :	19
Figure 4.1 - Get all Talent API	19
Postman Testing and Screenshots :	20
Figure 4.3 - Get all talents' details	20
Figure 4.4 - Get talent's details with id	20
Figure 4.5 - Search talent with keywords	21
Figure 4.6 - Using HTTPS	21
Figure 4.7 - Sequence Diagram	22
Task 5 :	22
Screenshots :	23
Task 6 :	26
Screenshots :	27
Figure 6.1 - Subscription Page	27
Figure 6.3 - Recurring Subscription	28
Figure 6.5 - Subscription status has been stored into local database	29
Figure 6.7 - Plan has been changed inside the database	30
Figure 6.9 - Subscription status has been updated	31
Figure 6.10 - Attempt to send a card which results failure of charging	31
Figure 6.11 - Charge status has been failed and recorded inside the database	32
Figure 6.12 - Sequence Diagram	32
Task 7 :	32
Screenshots :	33

Task 1 :



1. Invoking web service with
Jquery Ajax



2. Invoking web service with C#

Figure 1.1 - Difference between consuming web api with Jquery Ajax and C#

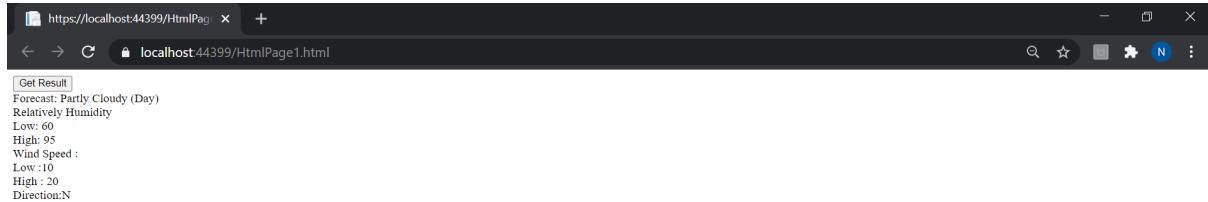


Figure 1.2 - Consuming web api with Ajax



Figure 1.3 - Consuming web api with C#

Task 2 :

API Documentation:

The screenshot shows the API documentation for the GET api/v3/products endpoint. It includes sections for Request Information, Response Information, and Response Formats.

Request Information

- URI Parameters: None
- Body Parameters: None

Response Information

- Resource Description: Collection of Product
- Table showing resource fields:

Name	Description	Type	Additional information
ID		integer	None
Name		string	Required
Category		string	None
Price		decimal number	Required Range: inclusive between 0 and 100

Response Formats

- application/json, text/json**:
Sample: [{ "id": 1, "name": "sample string 2", "category": "sample string 3", "price": 4.0 }, { "id": 2, "name": "sample string 2", "category": "sample string 3", "price": 4.0 }]
- application/xml, text/xml**:
Sample: <Product xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/AsTask2.Models"> <category>sample string 3</category> <name>sample string 2</Name> <price>4</Price>

Figure 2.1 - Get all product API

The screenshot shows the API documentation for the GET api/v3/products/{id} endpoint. It includes sections for Request Information, Response Information, and Response Formats.

Request Information

- URI Parameters: None
- Body Parameters: None

Response Information

- Resource Description: Product
- Table showing resource fields:

Name	Description	Type	Additional information
ID		integer	None
Name		string	Required
Category		string	None
Price		decimal number	Required Range: inclusive between 0 and 100

Response Formats

- application/json, text/json**:
Sample: { "id": 1, "name": "sample string 2", "category": "sample string 3", "price": 4.0 }
- application/xml, text/xml**:
Sample: <Product xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/AsTask2.Models"> <category>sample string 3</category> <name>sample string 2</Name> <price>4</Price>

Figure 2.2 - Get product with ID API

Help Page Home

GET api/v3/products?category=(category)

Request Information

URI Parameters

Name	Description	Type	Additional information
category		string	Required

Body Parameters

None

Response Information

Resource Description

Collection of Product

Name	Description	Type	Additional information
ID		integer	None
Name		string	Required
Category		string	None
Price		decimal number	Range: Inclusive between 0 and 100 Required

Response Formats

application/json, text/json

Sample:

```
{
  {
    "id": 1,
    "name": "sample string 2",
    "category": "sample string 3",
    "price": 4.0
  },
  {
    "id": 1,
    "name": "sample string 2",
    "category": "sample string 3",
    "price": 4.0
  }
}
```

application/xml, text/xml

Figure 2.3 - Get product with category API

Help Page Home

POST api/v3/products

Request Information

URI Parameters

None

Body Parameters

Product

Name	Description	Type	Additional information
ID		integer	None
Name		string	Required
Category		string	None
Price		decimal number	Required Range: Inclusive between 0 and 100

Request Formats

application/json, text/json

Sample:

```
{
  "id": 1,
  "name": "sample string 2",
  "category": "sample string 3",
  "price": 4.0
}
```

application/xml, text/xml

Sample:

```
<product xmlns:i1="http://www.w3.org/2001/XMLSchema-instance" xmlns:a="http://schemas.datacontract.org/2004/07/AssTask2.Models">
<i1:category>sample string 1</category>
<i1:id>
<i1:name>sample string 2</name>
<i1:price>4</price>
</product>
```

application/x-www-form-urlencoded

<https://localhost:44369/Help>

Figure 2.4 - Add new product API

Help Page Home

PUT api/v3/products/{id}

Request Information

URI Parameters

Name	Description	Type	Additional information
<code>id</code>		integer	Required

Body Parameters

Product	Description	Type	Additional information
<code>Name</code>		string	None
<code>id</code>		integer	Required
<code>Name</code>		string	Required
<code>Category</code>		string	None
<code>Price</code>		decimal number	Range inclusive between 0 and 100

Request Formats

application/json, text/json

Sample:

```
{
  "id": 1,
  "name": "sample string 2",
  "category": "sample string 3",
  "price": 4.0
}
```

application/xml, text/xml

Sample:

```
<product xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/Isstask1.Models">
  <category>sample string 3</category>
  <id>1</id>
  <name>sample string 2</name>
  <price>4</price>
</product>
```

application/x-www-form-urlencoded

Figure 2.5 - Update a product API

Help Page Home

DELETE api/v3/products/{id}

Request Information

URI Parameters

Name	Description	Type	Additional information
<code>id</code>		integer	Required

Body Parameters

None

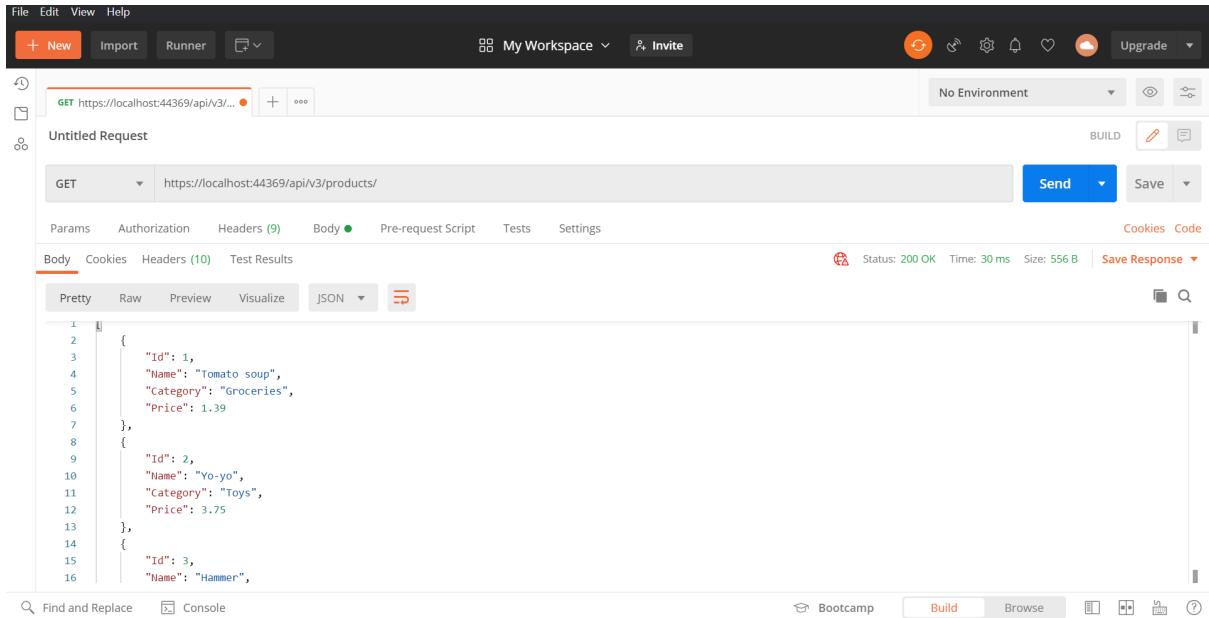
Response Information

Resource Description

None

Figure 2.6 - Delete product API

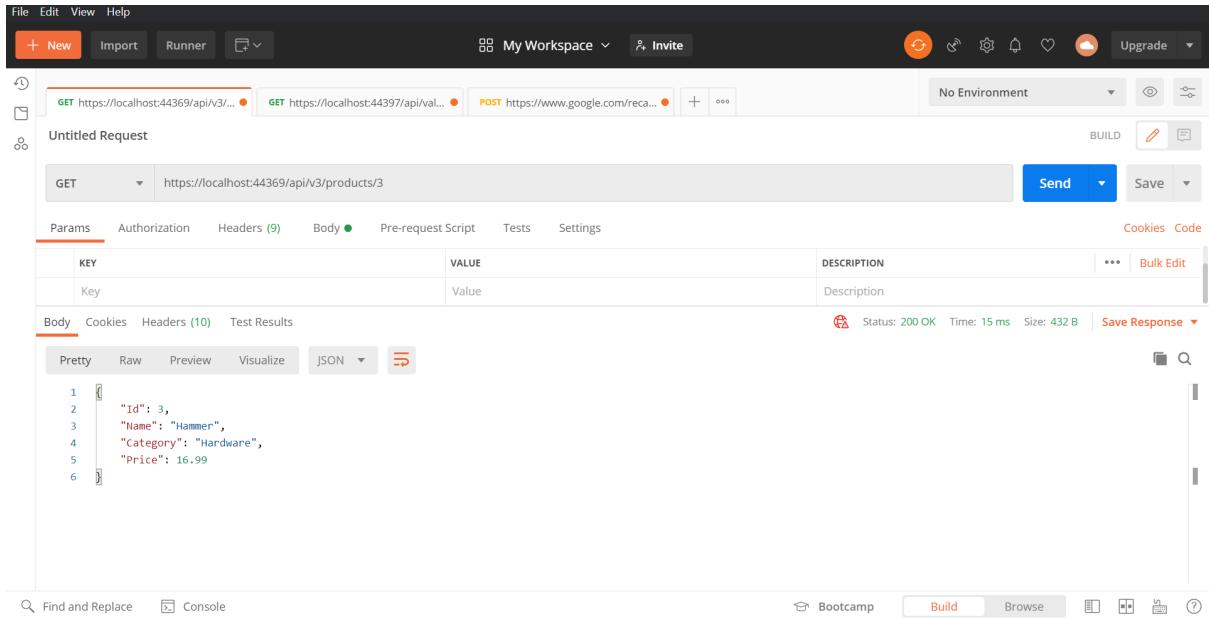
Postman Results and Screenshots:



The screenshot shows the Postman interface with a single request named "Untitled Request". The request method is GET, directed at <https://localhost:44369/api/v3/products>. The response status is 200 OK, time taken is 30 ms, and the size is 556 B. The response body is displayed in Pretty JSON format, showing a list of products with Id, Name, Category, and Price.

```
1 [
2   {
3     "Id": 1,
4     "Name": "Tomato soup",
5     "Category": "Groceries",
6     "Price": 1.39
7   },
8   {
9     "Id": 2,
10    "Name": "Yo-yo",
11    "Category": "Toys",
12    "Price": 3.75
13  },
14  {
15    "Id": 3,
16    "Name": "Hammer",
```

Figure 2.7 - Get all the product from application



The screenshot shows the Postman interface with a single request named "Untitled Request". The request method is GET, directed at <https://localhost:44369/api/v3/products/3>. The response status is 200 OK, time taken is 15 ms, and the size is 432 B. The response body is displayed in Pretty JSON format, showing a single product with Id, Name, Category, and Price.

```
1 [
2   {
3     "Id": 3,
4     "Name": "Hammer",
5     "Category": "Hardware",
6     "Price": 16.99
7   }
]
```

Figure 2.8 - Get Products by ID

The screenshot shows the Postman interface with an 'Untitled Request' collection. A GET request is made to `https://localhost:44369/api/v3/products?category=toys`. The response status is 200 OK, time is 16 ms, and size is 424 B. The response body is a JSON object:

```
1
2 {
3     "Id": 2,
4     "Name": "Yo-yo",
5     "Category": "Toys",
6     "Price": 3.75
7 }
```

Figure 2.9 - Get Products by Category

The screenshot shows the Postman interface with an 'Untitled Request' collection. A POST request is made to `https://localhost:44369/api/v3/products/`. The response status is 201 Created, time is 73 ms, and size is 484 B. The response body is a JSON object:

```
1
2 [
3     {
4         "Id": 4,
5         "Name": "haha",
6         "Category": "Toys",
7         "Price": 69.99
8     }
9 ]
```

Figure 2.10 - Create a product

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Invite'. Below the navigation is a workspace titled 'My Workspace' with an 'Invite' button. The main area is titled 'Untitled Request' and shows a 'PUT' request to 'https://localhost:44369/api/v3/products/4'. The 'Body' tab is selected, containing a table with three rows: 'Name' (Value: Basketball), 'Category' (Value: Sports), and 'Price' (Value: 100.00). Below the table are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'Text'. On the right, status information shows 'Status: 204 No Content', 'Time: 10 ms', and 'Size: 313 B'. At the bottom, there are buttons for 'Find and Replace' and 'Console'.

Figure 2.11.1 - A product has been updated

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', 'Help', 'New', 'Import', 'Runner', and 'Invite'. Below the navigation is a workspace titled 'My Workspace' with an 'Invite' button. The main area is titled 'Untitled Request' and shows a 'GET' request to 'https://localhost:44369/api/v3/products'. The 'Body' tab is selected, displaying a JSON response with two products. The first product has Id 3, Name "Hammer", Category "Hardware", and Price 3.75. The second product has Id 4, Name "Basketball", Category "Sports", and Price 100.00. Below the JSON is a code editor with line numbers 12 through 26. At the bottom, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. On the right, status information shows 'Status: 200 OK', 'Time: 10 ms', and 'Size: 616 B'. At the bottom, there are buttons for 'Find and Replace' and 'Console'.

Figure 2.11.2 - Result of updated product has been added to the product list successfully

The screenshot shows the Postman interface with the following details:

- Method:** DELETE
- URL:** https://localhost:44369/api/v3/products/4
- Status:** 204 No Content
- Time:** 32 ms
- Size:** 313 B

Figure 2.12 - Delete product from the list

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** https://localhost:44369/api/v3/products
- Body (JSON):**

```
1  [
2   "Id": 6,
3   "Name": "Basketball",
4   "Category": "Sports",
5   "Price": 100.00
6 ]
```

Figure 2.13 - Overposting

The screenshot shows the Postman application interface. At the top, there's a navigation bar with File, Edit, View, Help, + New, Import, Runner, and a workspace dropdown set to My Workspace. To the right are icons for Invite, Refresh, and Upgrade.

The main area is titled "Untitled Request". A POST request is being made to <https://localhost:44369/api/v3/products>. The "Body" tab is selected, showing the following JSON payload:

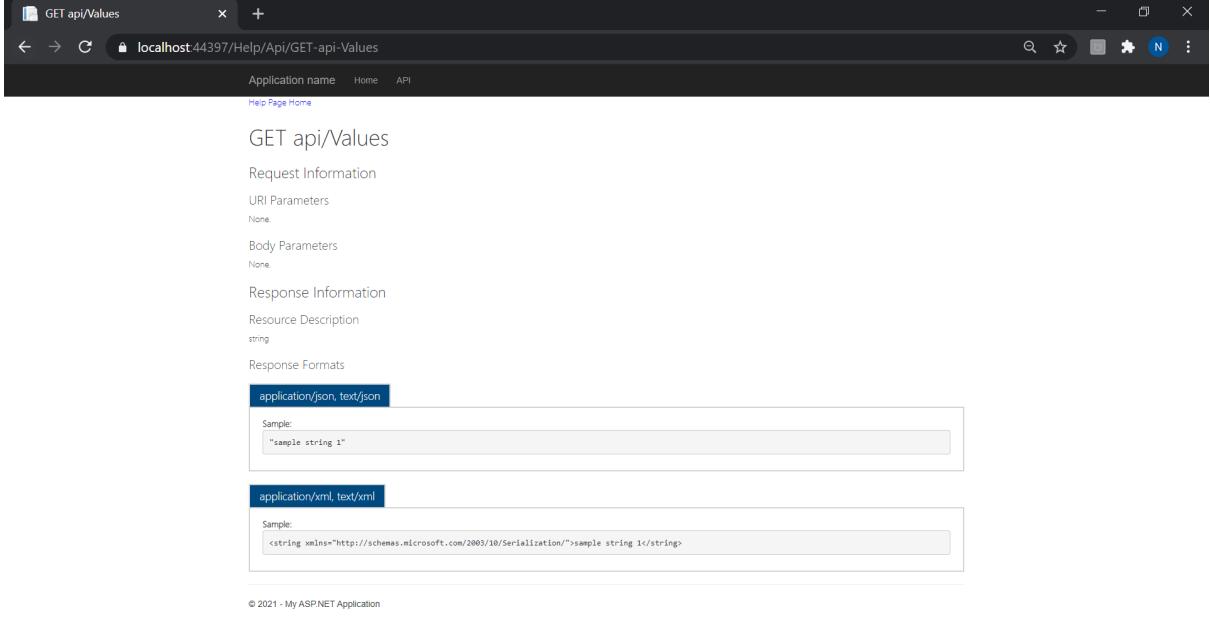
```
1 {  
2     "Name": "Basketball",  
3     "Category": "Sports",  
4     "item.Price": [  
5         "The Price property is required."  
6     ]  
}
```

Below the body, the status bar indicates Status: 400 Bad Request, Time: 251 ms, and Size: 479 B. The response pane shows the error message: "Message": "The request is invalid.", "ModelState": {"item.Price": ["The Price property is required."]}

Figure 2.14 - Underposting

Task 3:

API Documentation:



The screenshot shows a browser window displaying the API documentation for the `GET api/Values` endpoint. The URL in the address bar is `localhost:44397/Help/Api/GET-api-Values`. The page title is "GET api/Values". The content includes sections for Request Information, Response Information, and Response Formats.

Request Information

- URI Parameters: None.
- Body Parameters: None.

Response Information

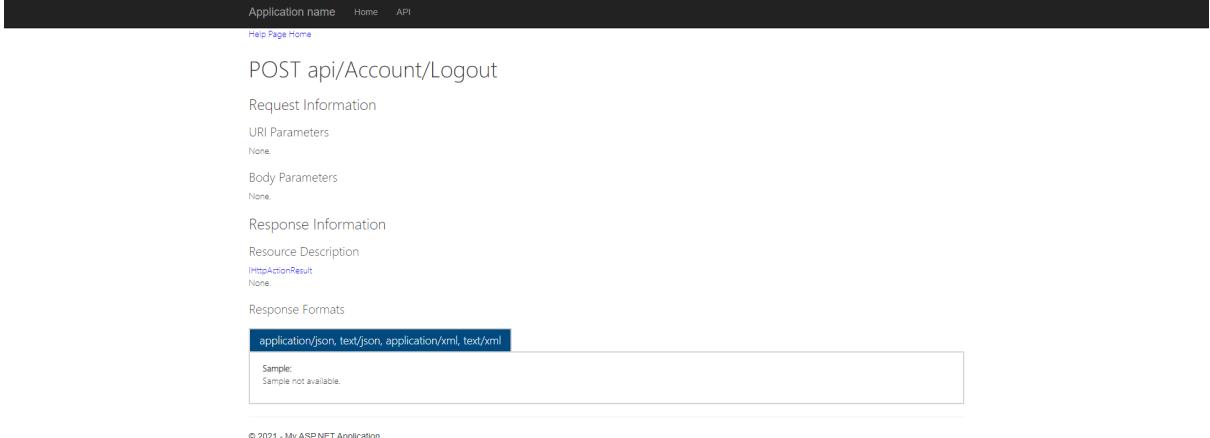
- Resource Description: string

Response Formats

- `application/json, text/json` (selected):
Sample:
"sample string 1"
- `application/xml, text/xml`:
Sample:
<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">sample string 1</string>

© 2021 - My ASP.NET Application

Figure 3.1 - Get Values API



The screenshot shows a browser window displaying the API documentation for the `POST api/Account/Logout` endpoint. The URL in the address bar is `localhost:44397/Help/Api/POST-api-Account-Logout`. The page title is "POST api/Account/Logout". The content includes sections for Request Information, Response Information, and Response Formats.

Request Information

- URI Parameters: None.
- Body Parameters: None.

Response Information

- Resource Description: `ActionResult`
None.

Response Formats

- `application/json, text/json, application/xml, text/xml` (selected):
Sample:
Sample not available.

© 2021 - My ASP.NET Application

Figure 3.2 - Logout Account API

POST api/Account/Register

Request Information

URI Parameters
None

Body Parameters

Name	Description	Type	Additional Information
Email		string	Required
Password		string	Required Data type: Password String length: inclusive between 6 and 100
ConfirmPassword		string	Data type: Password
GoogleCaptchaToken		string	None

Request Formats

application/json, text/json

Sample:

```
{
  "Email": "sample string 1",
  "Password": "sample string 2",
  "ConfirmPassword": "sample string 3",
  "GoogleCaptchaToken": "sample string 4"
}
```

Figure 3.3 - Register Account API

Postman Testing and Screenshot :

Untitled Request

POST https://localhost:44397/api/Account/Register

Params **Authorization** **Headers (9)** **Body** **Pre-request Script** **Tests** **Settings**

Body **Cookies** **Headers (9)** **Test Results**

KEY	VALUE	DESCRIPTION
Email	testingtesting@gmail.com	
Password	Testingtesting123!	
ConfirmPassword	Testingtesting123!	
GoogleCaptchaToken	03AGdBq24GnEPE3s51LuA_-XwEvwgnGQwvj6W9rnq63HM6bfqeVT...	

Status: 200 OK Time: 2.67 s Size: 328 B Save Response

Figure 3.4.1 - Register an user account with required params attached

The screenshot shows the Postman interface with an 'Untitled Request' collection. A POST request is made to <https://www.google.com/recaptcha/api/siteverify>. The 'Body' tab is selected, showing a JSON payload with 'response' and 'secret' fields. The response status is 200 OK, and the JSON body indicates success with a score of 0.9.

```

1 {
2   "success": true,
3   "challenge_ts": "2020-12-29T06:37:38Z",
4   "hostname": "localhost",
5   "score": 0.9,
6   "action": "register_account"
7 }
  
```

Figure 3.4.2 - The recaptcha verify api is called with GoogleCaptchaToken and secret key attached then a score will be returned to state whether the action is valid or not.

The screenshot shows the Postman interface with an 'Untitled Request' collection. A GET request is made to <https://localhost:44397/Token>. The 'Body' tab is selected, showing a JSON payload with 'grant_type', 'username', and 'password' fields. The response status is 200 OK, and the JSON body contains an access token.

```

1 "access_token": "zC4aQFkunu174gQF3InirDAy1t7XBSYhMKUWqax3JEFbPMTebgzEcEr3whsS2RX8eYCfu055xe1A3x196CUk8Mtmt1lWevAnnDGRQldfv4LT9WgIMXqVJEONa-vICXhXV931V-xLiuxSKZ7JQivE6XIGVZKaCeK54cpXzeB4qHOyTv5fh-xt8cuvqqKpEs_0fm6Hu13mN3R7T67-ti6xy0byHfvgSM4DCjXExwK_GUMmW1_0OfLbmQ7mYhXBslU8xC3XhDZfJURCHU-4WhSEUgqubliykZJcJ79T0Bb0zaxpRaiJArr5tQ3f9hQmu_SafSGxA07_CSFah1A_A1TxJ_Beszm7Qhu_UxwiULyCwRLd3BuLyrM9YChAwsvhgrwn-kuua60hybkgj4vsghtZbdCldhVDBXhJpsIPFccBAErw1Mj0m10xKUtbvtFagZ1p1Juc-fvjtUTB6AbVcA2U2obB8I_yH6GzL0GskccQVuSAO0H81hz_T9",
2 "token_type": "bearer"
  
```

Figure 3.5 - Login with the proper credentials

The screenshot shows the Postman interface with an 'Untitled Request' collection. A GET request is made to `https://localhost:44397/api/values`. In the 'Authorization' tab of the request settings, a 'Bearer Token' is selected. The response status is 200 OK, and the response body contains the message "Hello, testingtesting@gmail.com."

Figure 3.6 - Retrieve the result message with Bearer token attached in the request header

The screenshot shows the Postman interface with an 'Untitled Request' collection. A POST request is made to `https://localhost:44397/api/Account/Logout`. The 'Body' tab is selected, and the type is set to 'form-data'. The request body contains the following fields:

KEY	VALUE	DESCRIPTION
grant_type	password	
username	testingtesting@gmail.com	
password	Testingtesting123!	
GoogleCaptchaToken	03AGdBq24GnEPe3s51LuA_-XwEvwgnGQwvj6W9rnq63HM6bfqeVT...	

The response status is 200 OK.

Figure 3.7 - Logout from the application

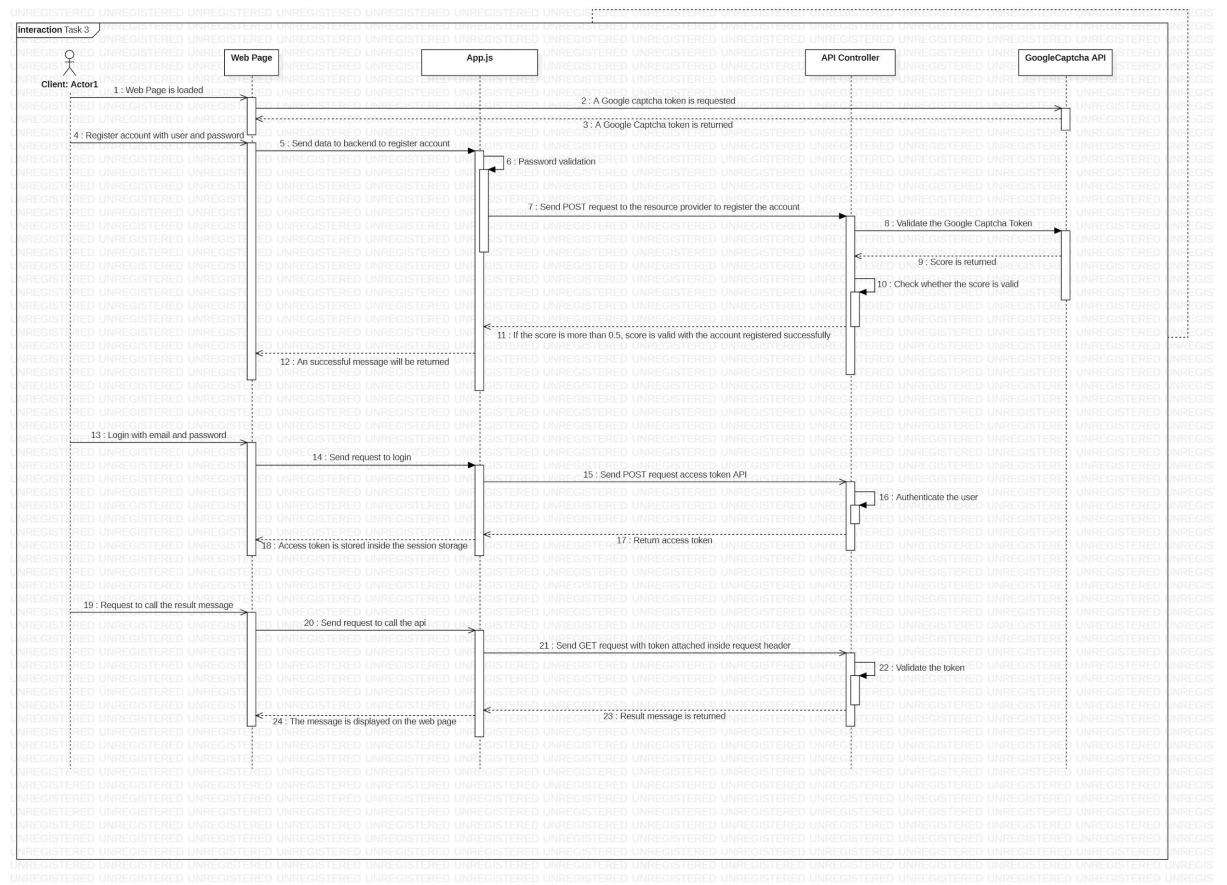


Figure 3.8 - Sequence Diagram

Task 4:

API Documentation :

The screenshot shows the API documentation for the `GET api/talents` endpoint. The browser title bar says "GET api/talents". The URL in the address bar is `localhost:44389/Help/Api/GET-api-talents`. The page content includes:

- Request Information:** None.
- Response Information:**
 - Resource Description:** Collection of `Talent`.
 - Table:** Shows the schema for the `Talent` collection.
- Response Formats:**
 - application/json, text/json:** Sample JSON response:

```
[{"id": 1, "name": "sample string 2", "shortname": "sample string 3", "biopic": "sample string 4"}, {"id": 2, "name": "sample string 2", "shortname": "sample string 3", "biopic": "sample string 4", "bio": "sample string 5"}]
```
 - application/xml, text/xml:** Sample XML response:

```
<talent xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/AssTask4.Models"> <i:nil> </talent>
```

Figure 4.1 - Get all Talent API

The screenshot shows the API documentation for the `GET api/talents/{id}` endpoint. The browser title bar says "GET api/talents/{id}". The URL in the address bar is `localhost:44389/Help/Api/GET-api-talents-id`. The page content includes:

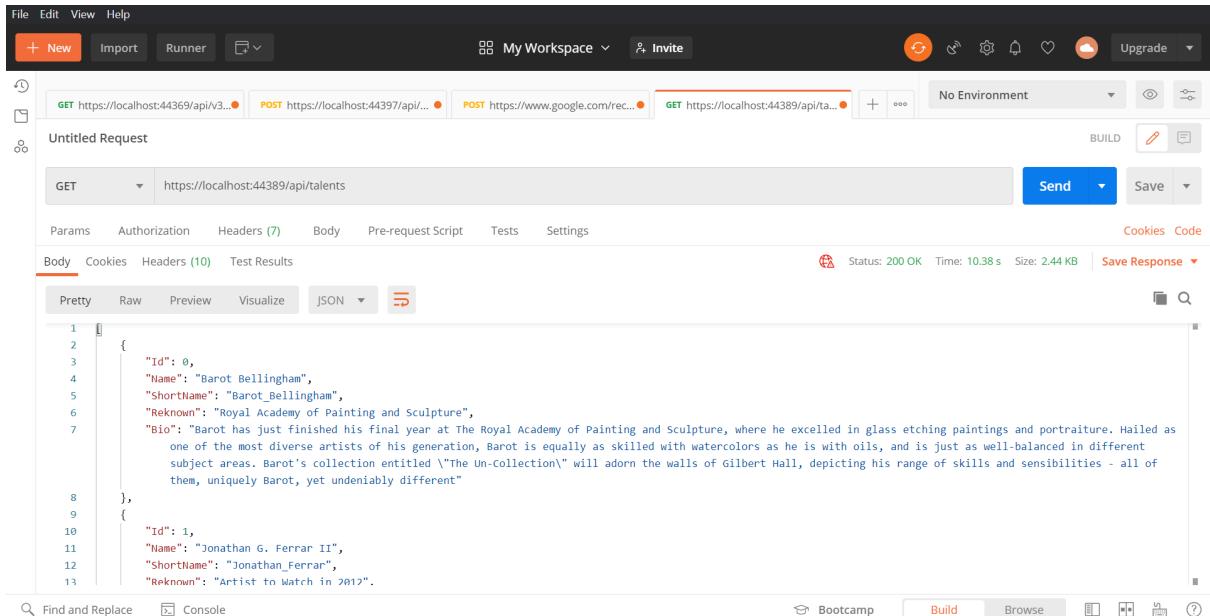
- Request Information:** None.
- Response Information:**
 - Resource Description:** `Talent`.
 - Table:** Shows the schema for the `Talent` entity.
- Response Formats:**
 - application/json, text/json:** Sample JSON response:

```
{"id": 1, "name": "sample string 2", "shortname": "sample string 3", "biopic": "sample string 4", "bio": "sample string 5"}
```
 - application/xml, text/xml:** Sample XML response:

```
<talent xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/AssTask4.Models"> <i:nil> </talent>
```

Figure 4.2 - Get Talent with ID

Postman Testing and Screenshots :



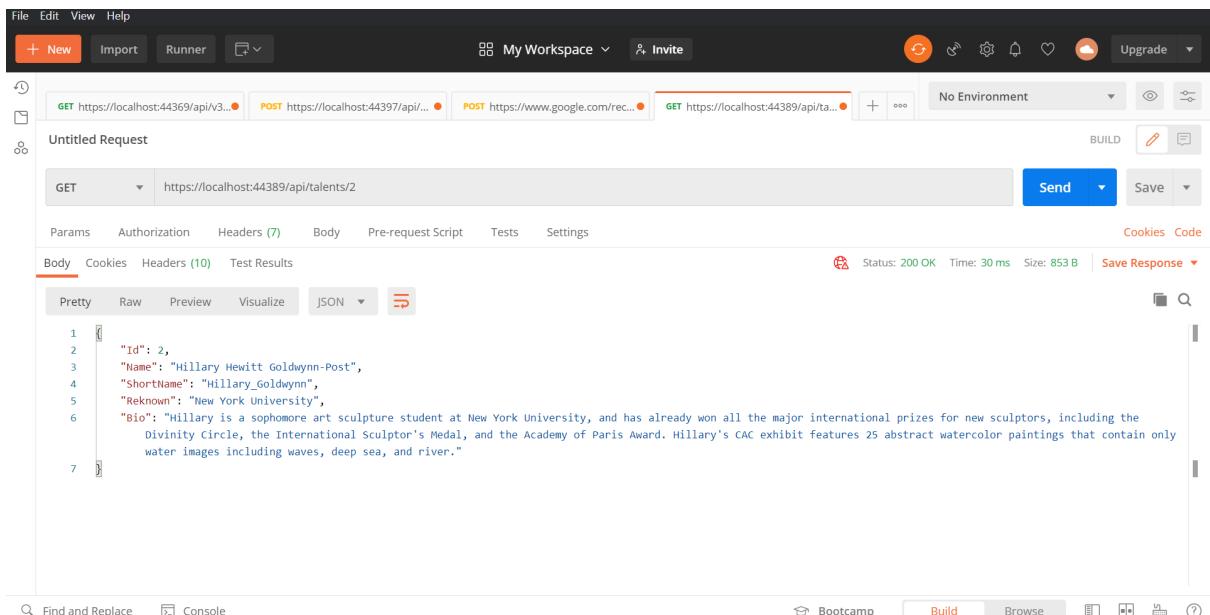
The screenshot shows the Postman interface with the following details:

- Header Bar:** File, Edit, View, Help, + New, Import, Runner, My Workspace, Invite, No Environment, Upgrade.
- Request List:** GET https://localhost:44369/api/v3..., POST https://localhost:44397/api/..., POST https://www.google.com/rec..., GET https://localhost:44389/api/ta... (selected).
- Current Request:** Untitled Request, GET https://localhost:44389/api/talents.
- Request Details:** Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings, Body tab selected.
- Response Preview:** Status: 200 OK, Time: 10.38 s, Size: 2.44 KB. Response body is a JSON array of talents:

```
1  [
2    {
3      "Id": 0,
4      "Name": "Barot_Bellingham",
5      "Shortname": "Barot_Bellingham",
6      "Rekknown": "Royal Academy of Painting and Sculpture",
7      "Bio": "Barot has just finished his final year at The Royal Academy of Painting and Sculpture, where he excelled in glass etching paintings and portraiture. Hailed as one of the most diverse artists of his generation, Barot is equally as skilled with watercolors as he is with oils, and is just as well-balanced in different subject areas. Barot's collection entitled \"The Un-collection\" will adorn the walls of Gilbert Hall, depicting his range of skills and sensibilities - all of them, uniquely Barot, yet undeniably different"
8    },
9    {
10      "Id": 1,
11      "Name": "Jonathan_G_Ferrar_II",
12      "Shortname": "Jonathan_Ferrar",
13      "Rekknown": "Artist to Watch in 2012".
```

- Bottom Navigation:** Find and Replace, Console, Bootcamp, Build, Browse.

Figure 4.3 - Get all talents' details



The screenshot shows the Postman interface with the following details:

- Header Bar:** File, Edit, View, Help, + New, Import, Runner, My Workspace, Invite, No Environment, Upgrade.
- Request List:** GET https://localhost:44369/api/v3..., POST https://localhost:44397/api/..., POST https://www.google.com/rec..., GET https://localhost:44389/api/ta... (selected).
- Current Request:** Untitled Request, GET https://localhost:44389/api/talents/2.
- Request Details:** Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings, Body tab selected.
- Response Preview:** Status: 200 OK, Time: 30 ms, Size: 853 B. Response body is a JSON object:

```
1  {
2    "Id": 2,
3    "Name": "Hillary Hewitt Goldwynn-Post",
4    "Shortname": "Hillary_Goldwynn",
5    "Rekknown": "New York University",
6    "Bio": "Hillary is a sophomore art sculpture student at New York University, and has already won all the major international prizes for new sculptors, including the Divinity Circle, the International Sculptor's Medal, and the Academy of Paris Award. Hillary's CAC exhibit features 25 abstract watercolor paintings that contain only water images including waves, deep sea, and river."
```

- Bottom Navigation:** Find and Replace, Console, Bootcamp, Build, Browse.

Figure 4.4 - Get talent's details with id

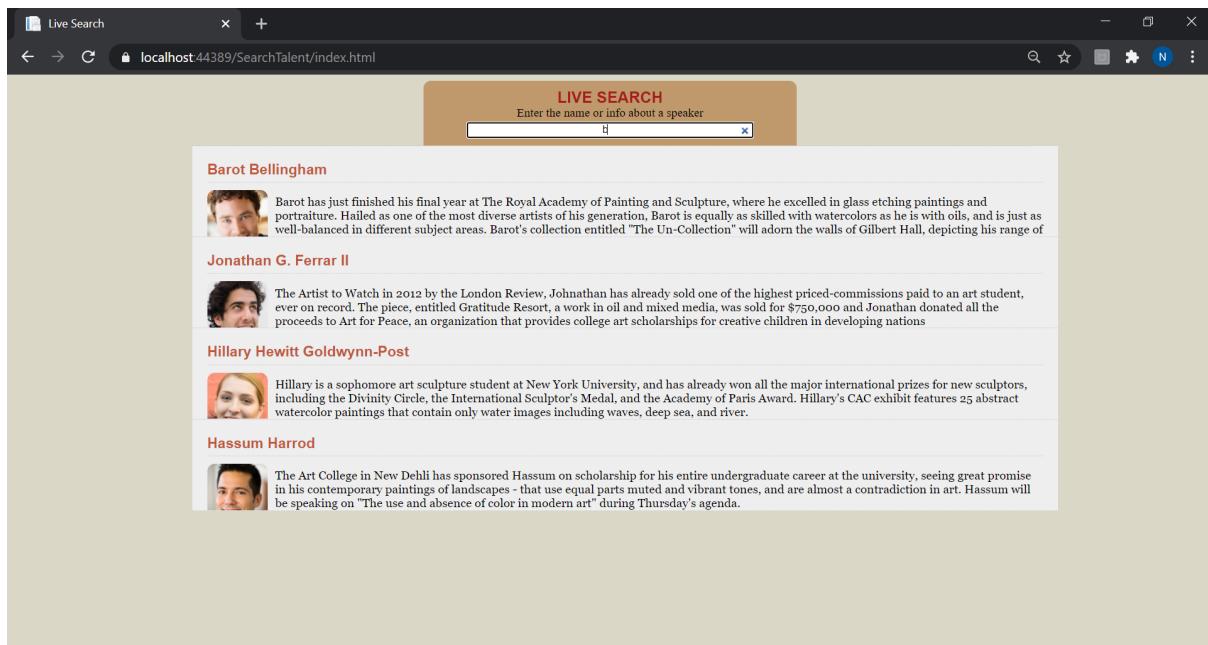


Figure 4.5 - Search talent with keywords

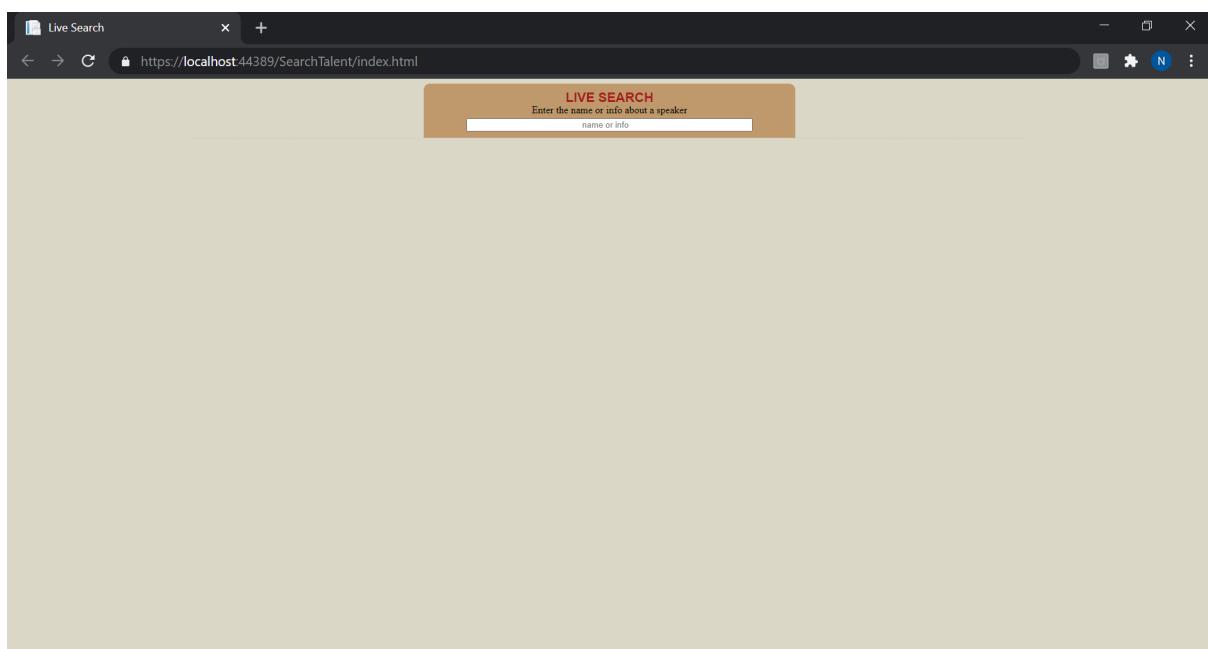


Figure 4.6 - Using HTTPS

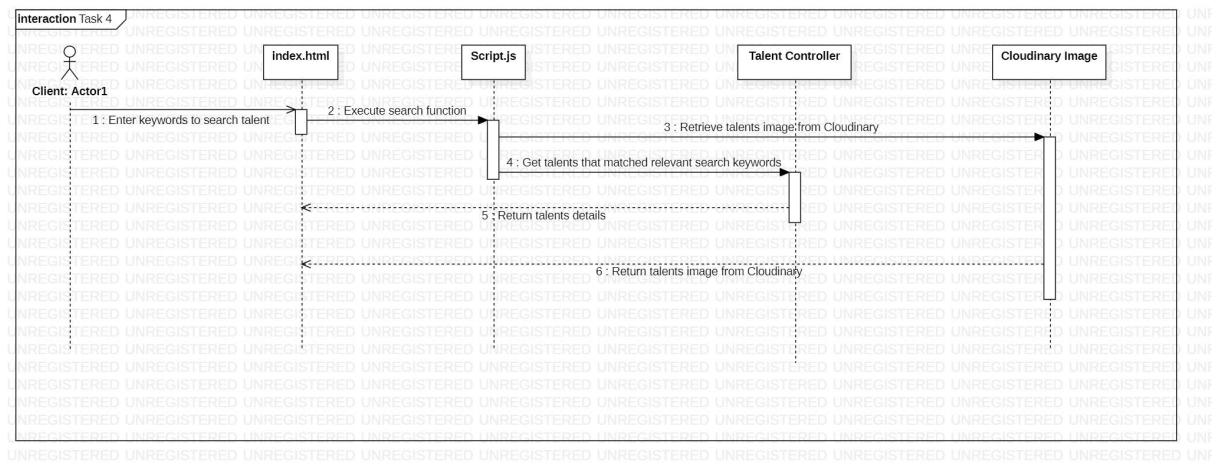


Figure 4.7 - Sequence Diagram

Task 5 :

Screenshots :



Figure 5.1 - Upload to S3 Page

Steps :

- 1) Upload the file by clicking that “Choose File” button.
- 2) Click on the “Upload to AWS S3” button to upload the images to AWS S3.
- 3) And the progress bar will show the request progress.
- 4) Additional : Validation occurs before uploading the file to AWS S3.
Only Images with format (.JPEG/.JPG/.PNG) are accepted.

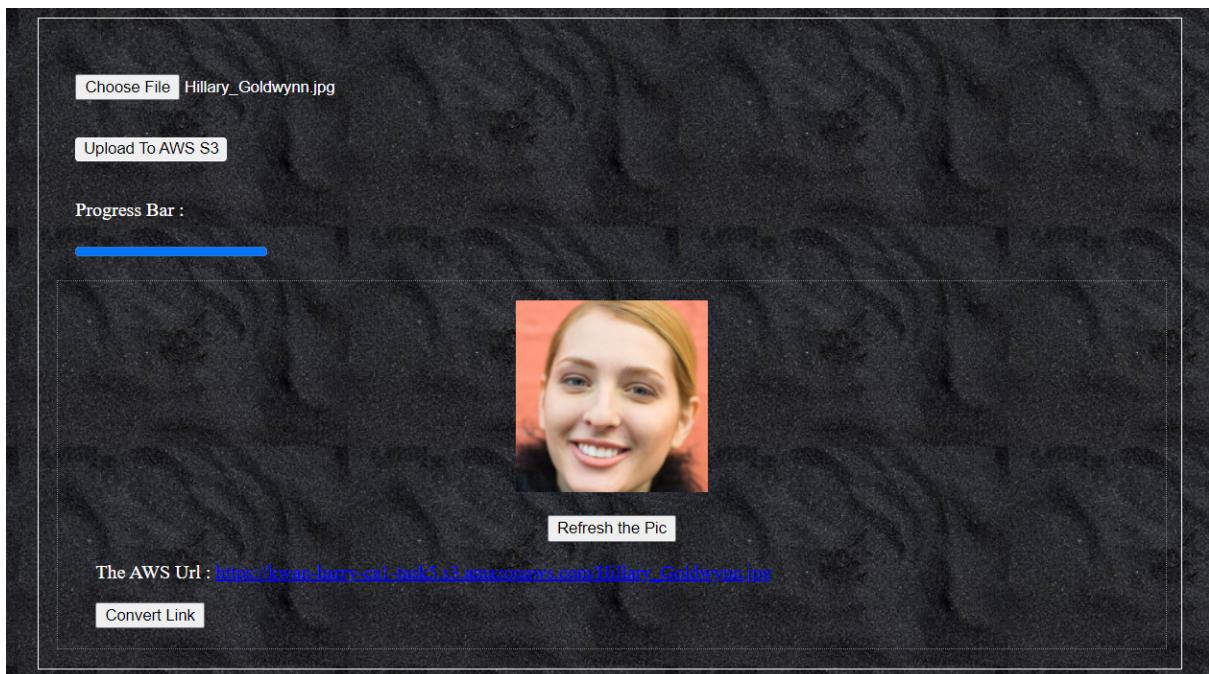


Figure 5.2 - Showing result

Steps :

- 1) Click on the “Refresh the Pic” button if the picture is not loaded.
- 2) The AWS Url is shown at the bottom.
- 3) Click on the “Convert Link” button to convert the long AWS Url to a short Url.

- 4) Can click on the href link to check whether the image is uploaded and stored successfully in AWS S3.

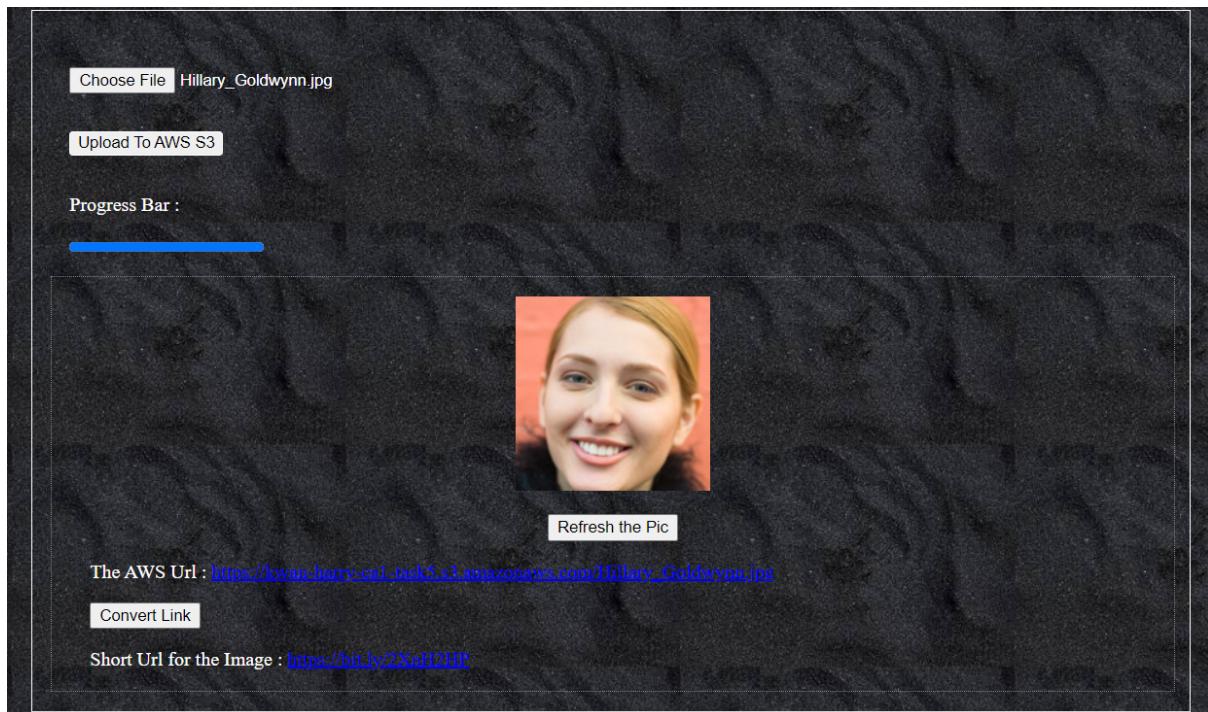


Figure 5.3 - Shorten the AWS Url

Step :

- 1) Click on the link to check whether the shorter link is working well.

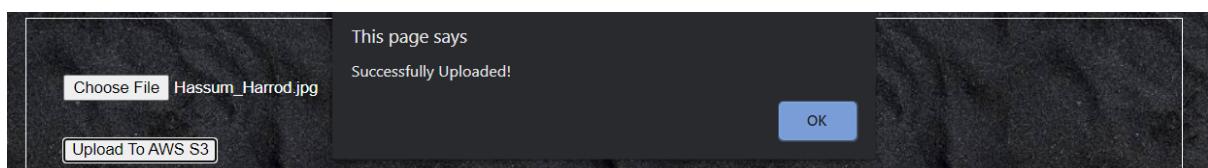


Figure 5.4 - Upload Successfully Message shown

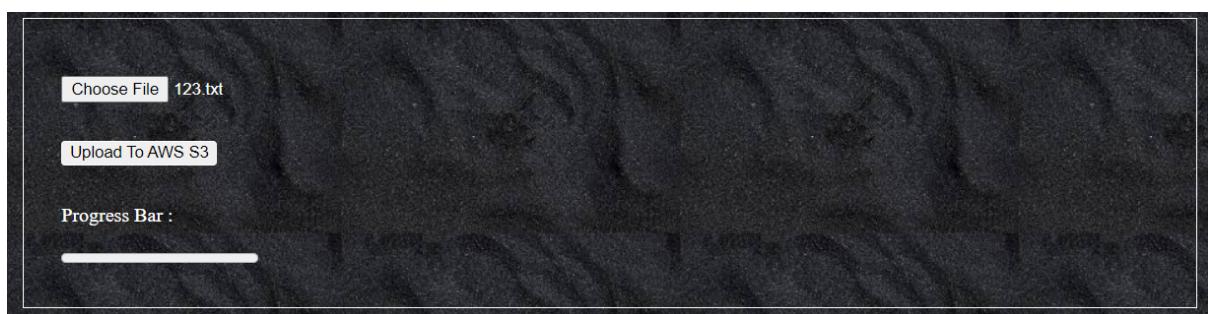


Figure 5.5 - File Validation

Step :

- 1) Upload the file that the file type is not supported.

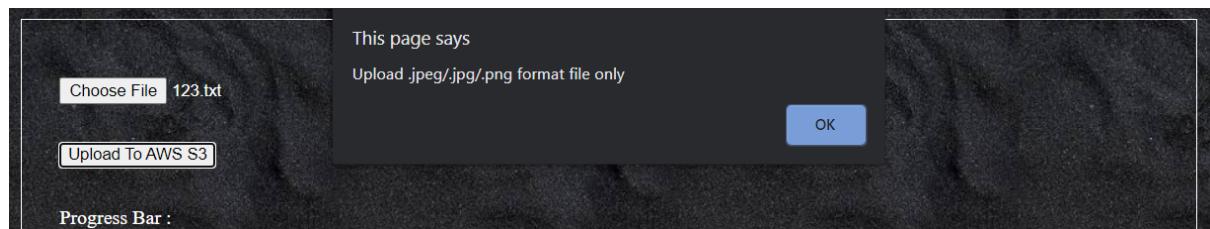


Figure 5.6 Alert message shown.



Figure 5.7 Error message shown.



Figure 5.8 Prevent uploading empty file

-The “Upload to AWS S3” button is disabled when there is no file selected.

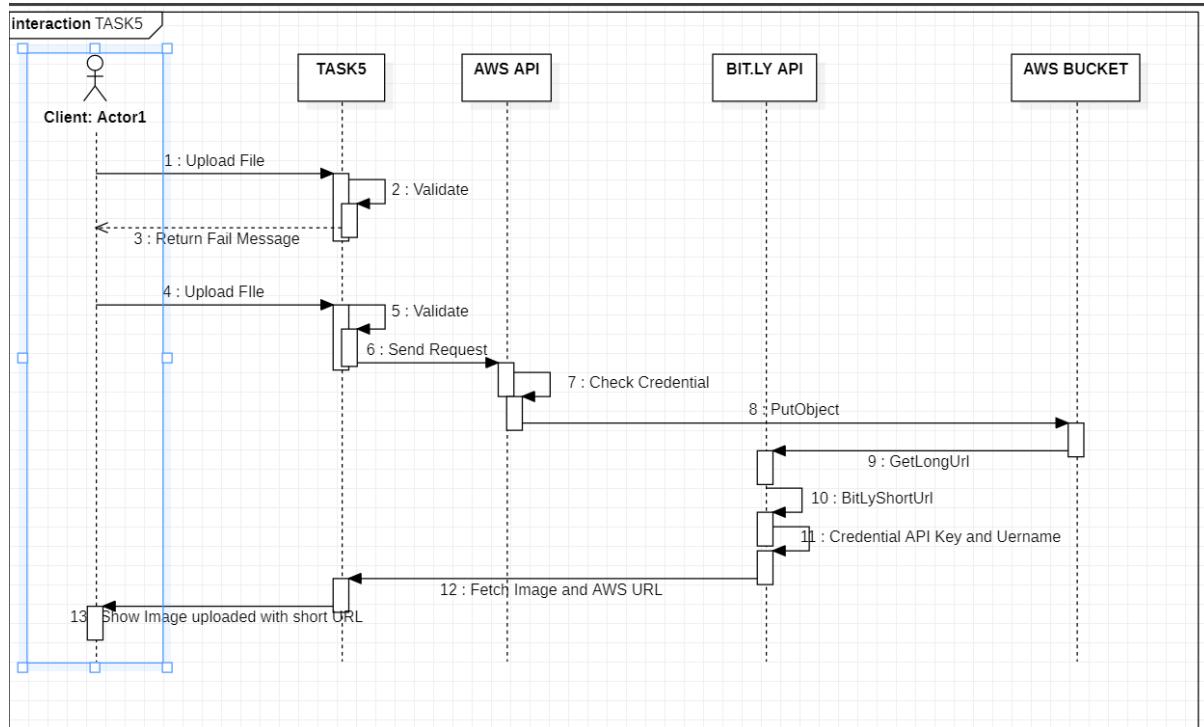


Figure 5.9 Sequence Diagram

Task 6 :

Screenshots :

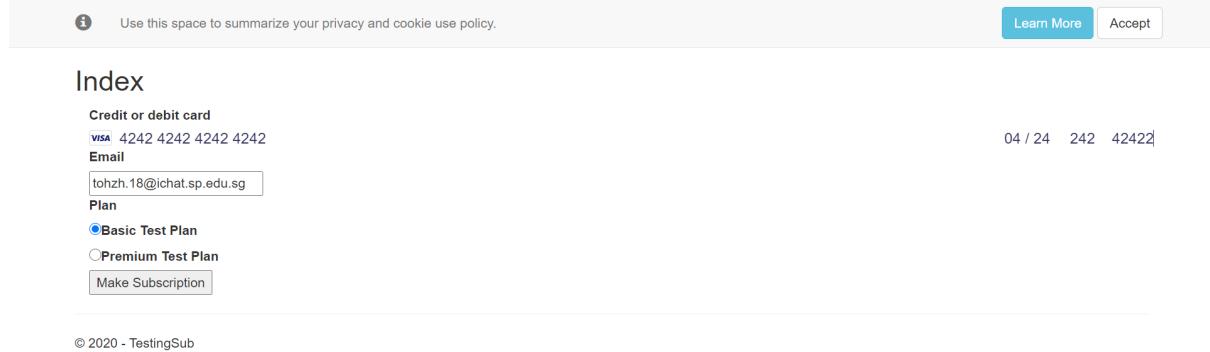


Figure 6.1 - Subscription Page

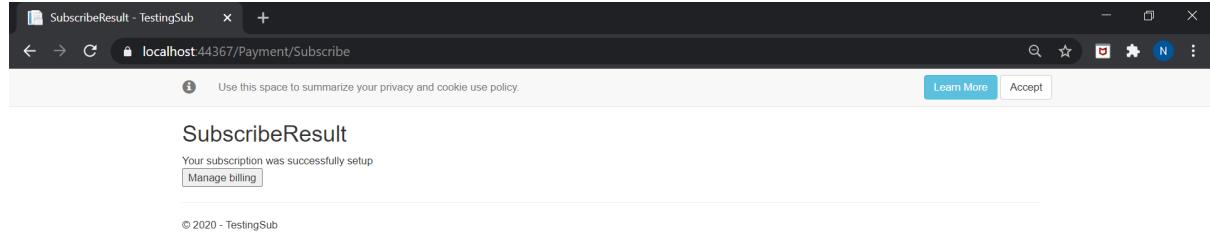


Figure 6.2 - Subscribe Result

The screenshot shows the Stripe dashboard for a user named 'harry'. The left sidebar includes links for Home, Activate your account, Payments, Balances, Customers (selected), Connected accounts, Products, Reports, Developers, Viewing test data, and Settings. The main area displays a customer profile for 'tohzhi.18' (@ichat.sp.edu.sg) with the name 'Name not provided'. The customer has spent SGD 320.00 since Dec 2020, with an MRR of SGD 304.37. A 'Subscriptions' section shows a single active subscription for 'Test Product' with a daily billing cycle, set to renew on Jan 6 for SGD 10.00. Below this, a 'Payments' section lists several successful SGD 10.00 transactions from January 2020 to December 2020. A 'Details' panel on the left provides account and billing information.

Figure 6.3 - Recurring Subscription

The screenshot shows the Stripe dashboard for a user named 'harry'. The left sidebar includes links for Home, Activate your account, Payments, Balances, Customers, Connected accounts, Products, Reports, Developers, API keys, Webhooks (selected), Events, Logs, Viewing test data, and Settings. The main area displays a 'Webhook attempts' section titled 'Attempts to send an event to your endpoint in the past 15 days'. It shows a table with columns for EVENT TYPE, EVENT ID, CREATED, and NEXT RETRY. All attempts listed are successful (Succeeded). The table includes entries for invoice.created, charge.succeeded, and various other events like customer.subscription.updated. A total of 271 results are shown, with 'Next' and 'Previous' buttons at the bottom.

Figure 6.4 - Webhook attempts to application

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'CustomerInfo' is selected under 'LAPTOP-0NC7NGDU\SQLEXPRESS'. In the center pane, a query window displays the following T-SQL code:

```

SELECT TOP (1000) [customer_id]
      ,[email]
      ,[subscription_status]
      ,[charge_status]
      ,[basic]
      ,[premium]
  FROM [CustomerInfo].[dbo].[Customer]

```

The results grid below shows one row of data:

customer_id	email	subscription_status	charge_status	basic	premium
cus_1hx7zUMyVTCfZ	tohzh.18@ichat.sp.edu.sg	active	succed	True	False

At the bottom of the results grid, a message states: "Query executed successfully." Below the grid, status information is displayed: Ln 8, Col 39, Ch 39, and INS.

Figure 6.5 - Subscription status has been stored into local database

The screenshot shows a web browser displaying a Stripe confirmation page titled "Confirm your new plan". The page is for a user named "harry" in "Test mode". The headline says "Testing Headline" and includes a link to "Return to harry". The main content area shows a "Premium Test Plan" starting on January 6, 2021, with a daily payment of SGD 20.00. It also shows an amount due today of SGD 10.00, paid via Visa ending in 4242. A large blue "Confirm" button is at the bottom, and a "Go back" button is below it. A small note at the bottom states: "By confirming your new plan, you agree to harry's Terms of Service and Privacy Policy."

Figure 6.6 - Change plan

```

SELECT TOP (1000) [customer_id]
      ,[email]
      ,[subscription_status]
      ,[charge_status]
      ,[basic]
      ,[premium]
   FROM [CustomerInfo].[dbo].[Customer]

```

	customer_id	email	subscription_status	charge_status	basic	premium
1	cus_lhx7z2uMyVTCZ	tohzh.18@ichat.sp.edu.sg	active	succeed	False	True

Query executed successfully.

Figure 6.7 - Plan has been changed inside the database

Testing Headline

← Return to harry

Cancel your plan

CURRENT PLAN

Test Product
SGD 10.00 per day

If you cancel this plan, it will no longer be available to you.

Cancel plan

Go back

By cancelling your plan, you agree to harry's [Terms of Service](#) and [Privacy Policy](#).

Powered by **stripe** | [Terms](#) [Privacy](#)

Figure 6.8 - Plan cancellation

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'CustomerInfo' is selected under 'LAPTOP-0NC7NGDU\SQLEXPRESS'. A query window titled 'SQLQuery1.sql - LAPTOP-0NC7NGDU\user (54) - Microsoft SQL Server Management Studio' contains the following SQL code:

```

SELECT TOP (1000) [customer_id]
      ,[email]
      ,[subscription_status]
      ,[charge_status]
      ,[basic]
      ,[premium]
   FROM [CustomerInfo].[dbo].[Customer]

```

The results pane shows one row of data:

	customer_id	email	subscription_status	charge_status	basic	premium
1	cus_1nx7zZuMyVTCtZ	tohzh.18@chat.sp.edu.sg	inactive	succeed	False	True

At the bottom, a message bar indicates: 'Query executed successfully.' with details: 'Ln 1 Col 1 Ch 1 INS' and 'LAPTOP-0NC7NGDU\SQLEXPRESS ... LAPTOP-0NC7NGDU\user (54) CustomerInfo 00:00:00 1 rows'.

Figure 6.9 - Subscription status has been updated

The screenshot shows a web-based form for creating a card subscription. At the top, there is a message: 'Use this space to summarize your privacy and cookie use policy.' with 'Learn More' and 'Accept' buttons. Below this, the form fields are:

- Credit or debit card**: A placeholder text field containing 'visa 4000 0000 0000 0341'.
- Email**: An input field containing 'testfailed@gmail.com'.
- Plan**: A radio button group with two options: 'Basic Test Plan' (selected) and 'Premium Test Plan'.
- Make Subscription**: A submit button.

At the bottom left, there is a copyright notice: '© 2020 - TestingSub'.

Figure 6.10 - Attempt to send a card which results failure of charging

```

SELECT TOP (1000) [customer_id]
      ,[email]
      ,[subscription_status]
      ,[charge_status]
      ,[basic]
      ,[premium]
  FROM [CustomerInfo].[dbo].[Customer]

```

	customer_id	email	subscription_status	charge_status	basic	premium
1	cus_1lSrKp5y0JSwAO	testfailed@gmail.com	incomplete	failed	True	False

Query executed successfully.

Figure 6.11 - Charge status has been failed and recorded inside the database

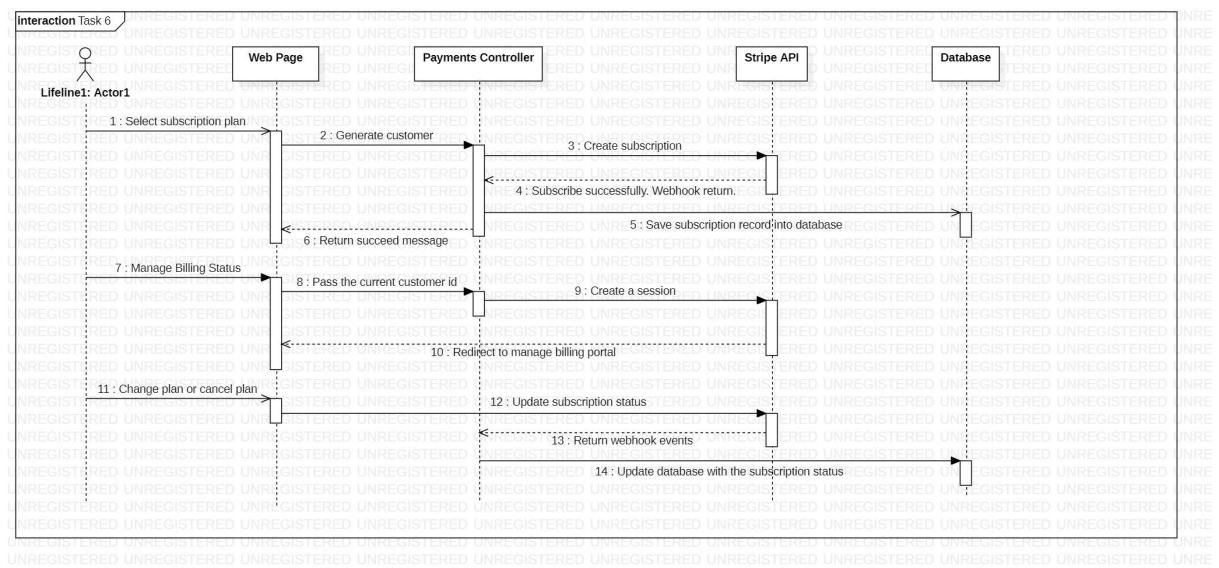


Figure 6.12 - Sequence Diagram

Task 7 :

Screenshots :

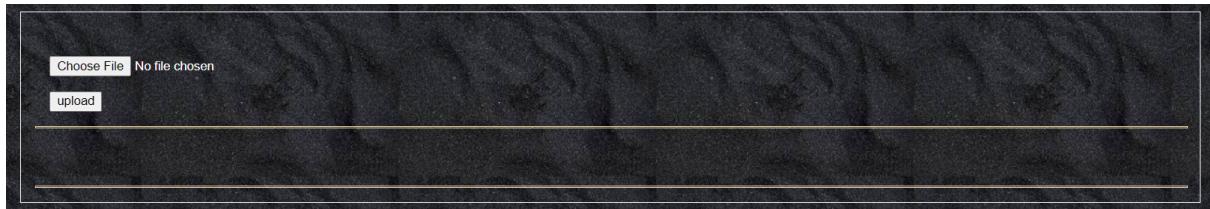


Figure 7.1 - Task 7 Page

Steps :

- 1) Upload the File by clicking “Choose File” button.
- 2) Click on the “Upload” button to recognise the image.

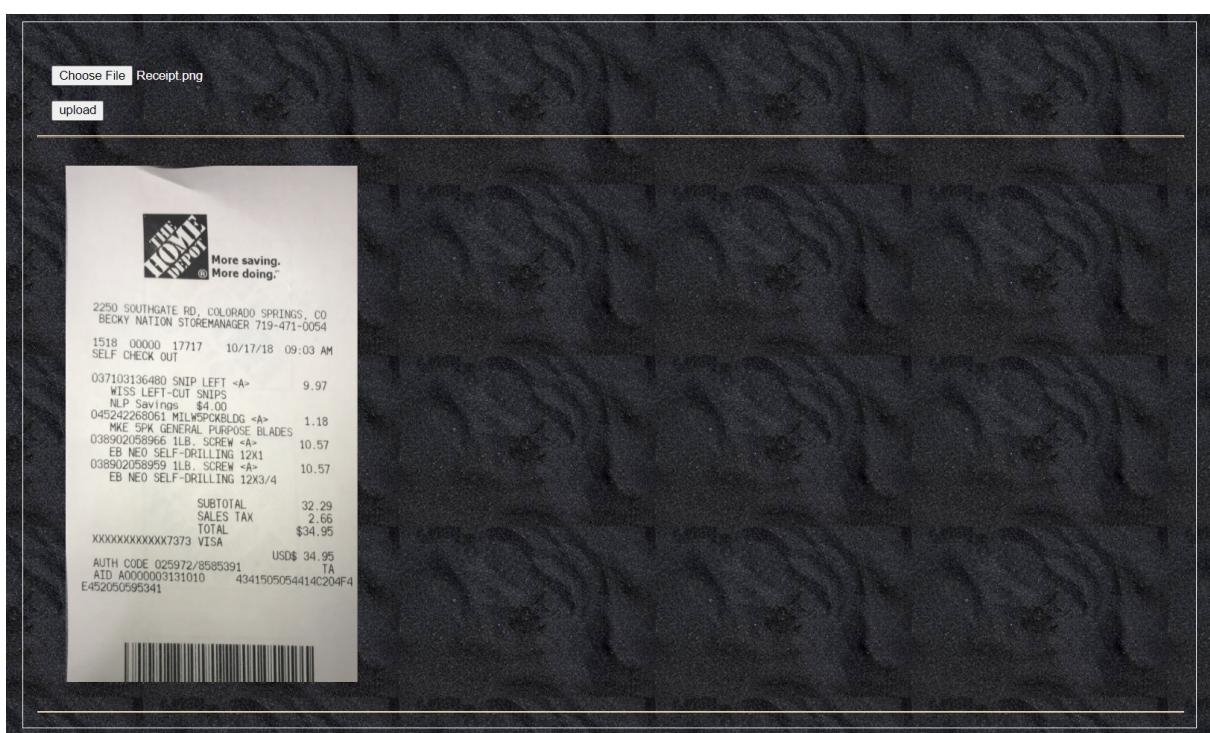


Figure 7.2 - Image Preview in the Table



Figure 7.3 - Loading Gif while processing the request

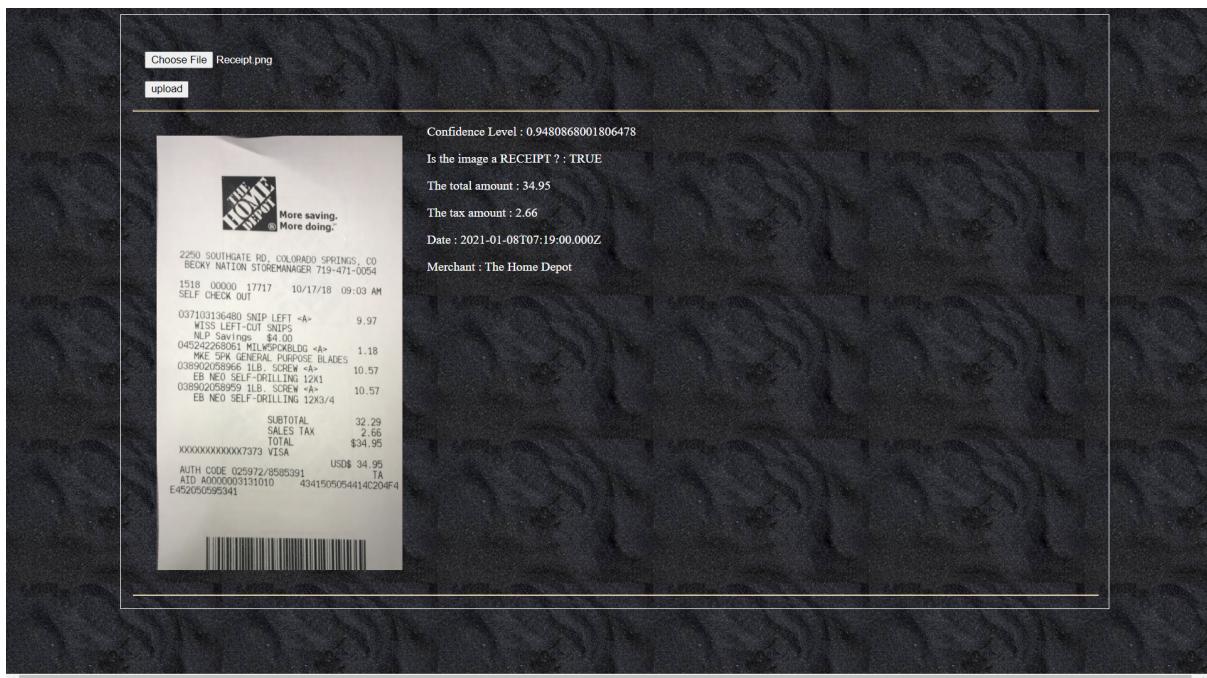


Figure 7.4 - The information of the image shown (receipt image)

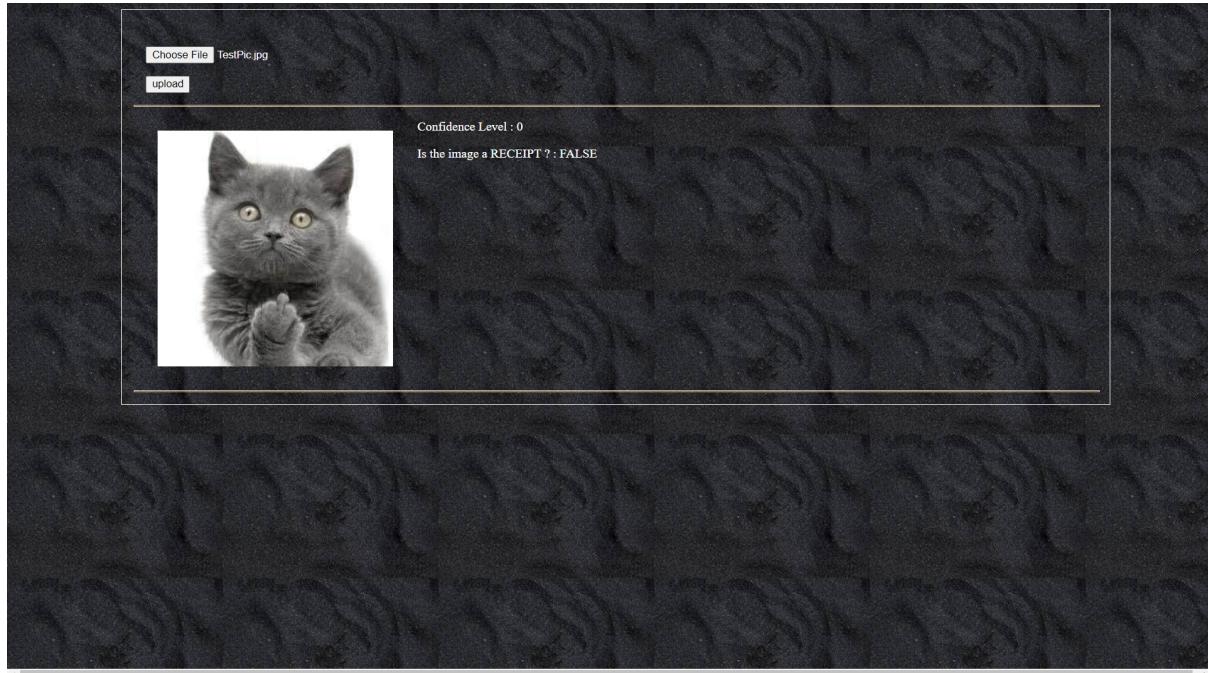


Figure 7.5 - The information of the image shown (not a receipt image)

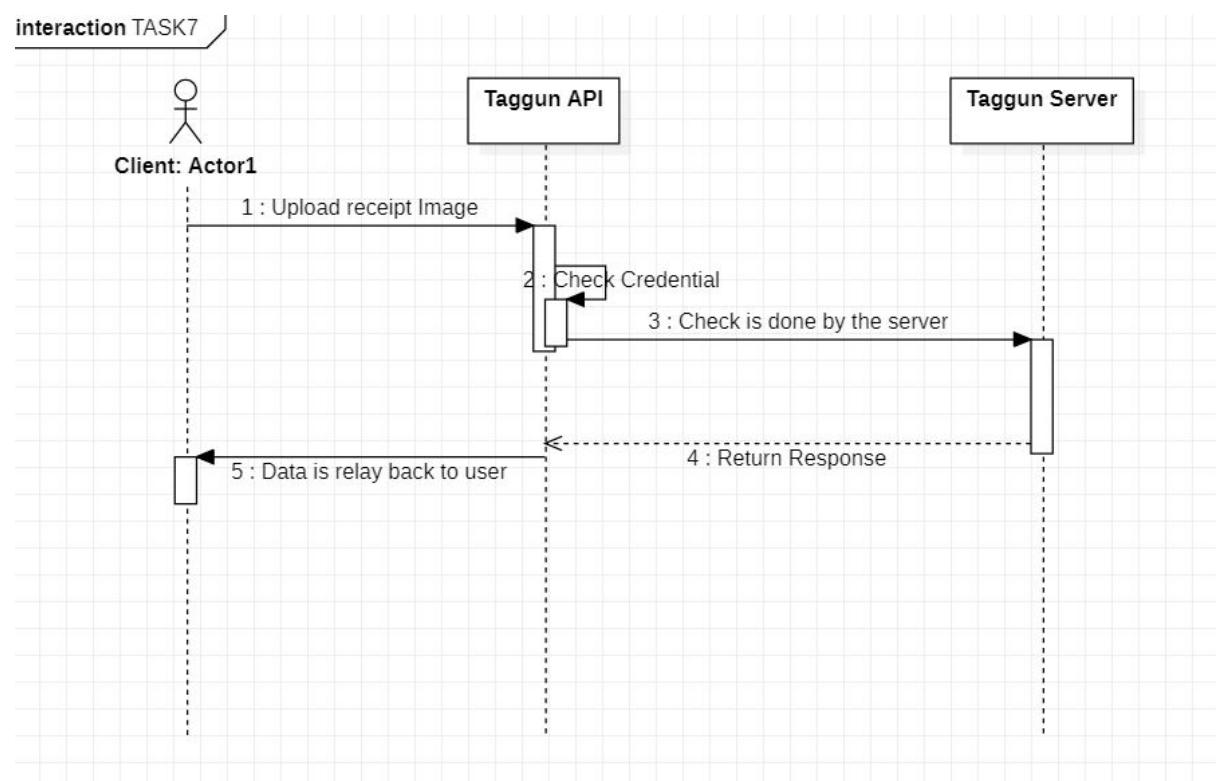


Figure 7.6 - Sequence Diagram