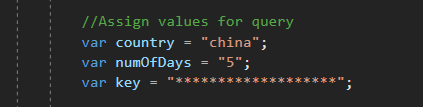
CSC Assignment 1 Documentation

Task 1

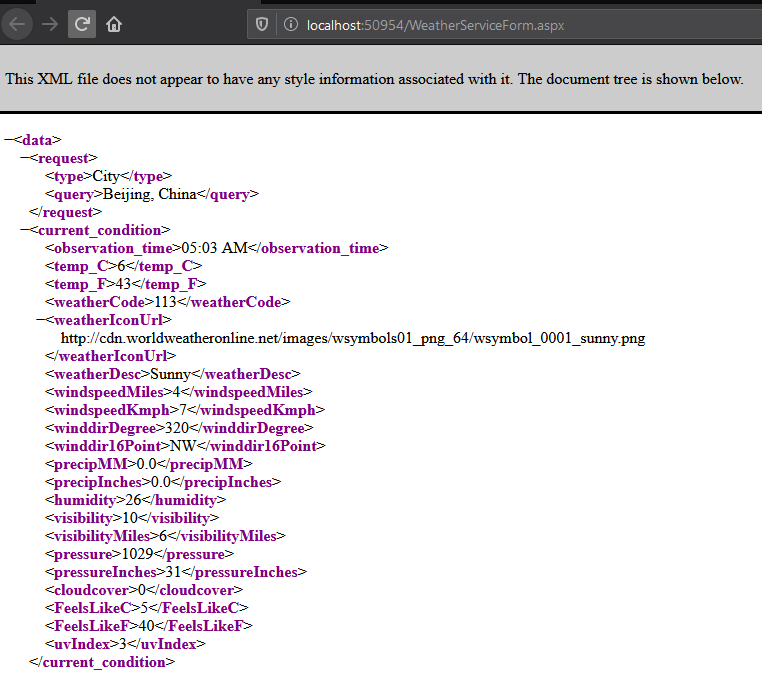
Browser (aspx)

*Located in WeatherService Folder*

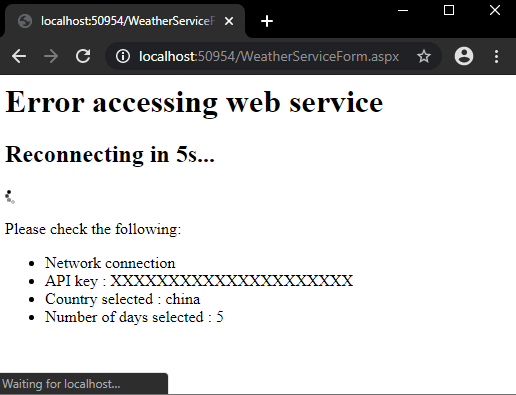
Assign the country, number of days and insert your key in WeatherServiceForm.aspx.cs from <https://www.worldweatheronline.com/developer/> .



Debug run the application and you will see the following page on load.



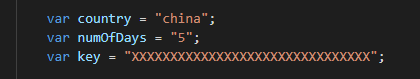
In the case of error, the web service will ask u to verify your API key as well as your condition inputs.



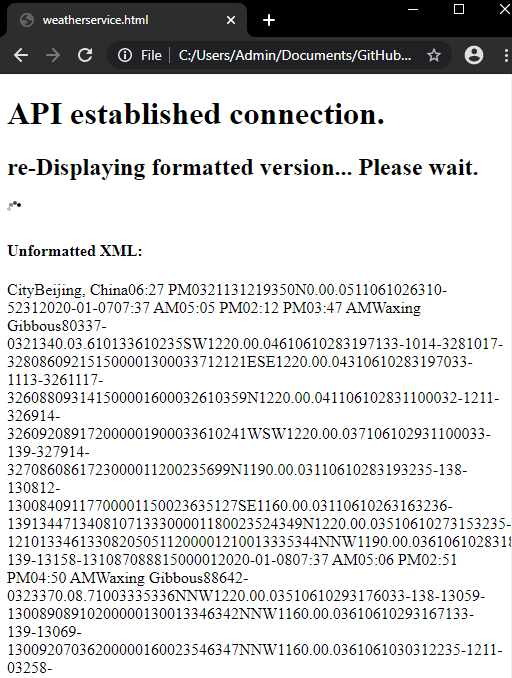
Browser (JQuery)

*Located in WeatherService\_js Folder*

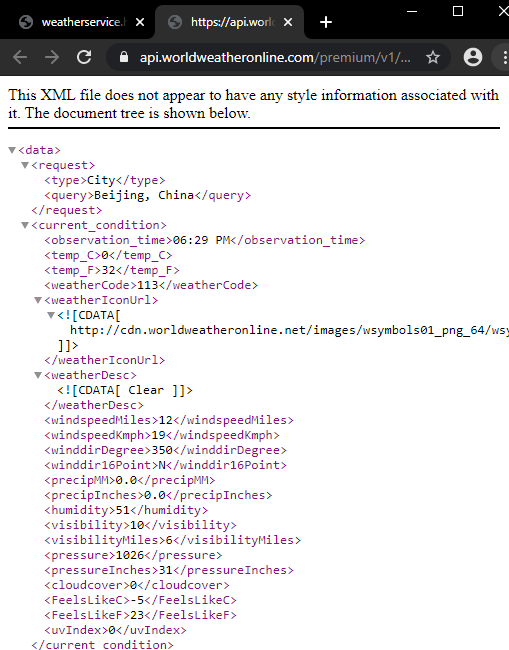
Assign the country, number of days and insert your key in script.js from <https://www.worldweatheronline.com/developer/> .



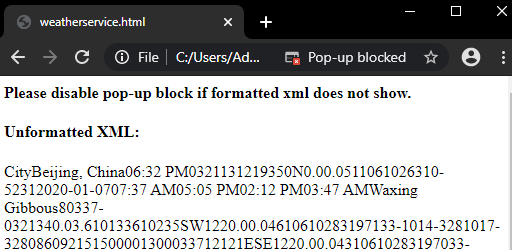
Debug run the application and you will see the following page on API successful connection.



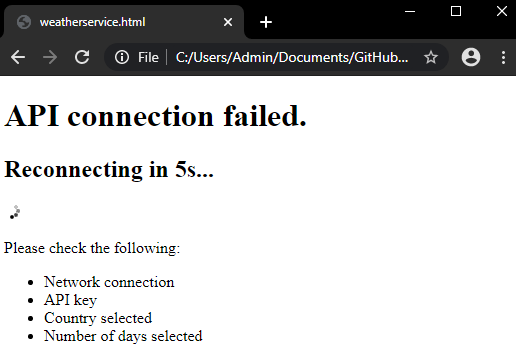
A formatted XML will be pop up in a new window. (Direct XML file from world weather)



In the case of formatted XML not popping up, enable or allow pop-up window.

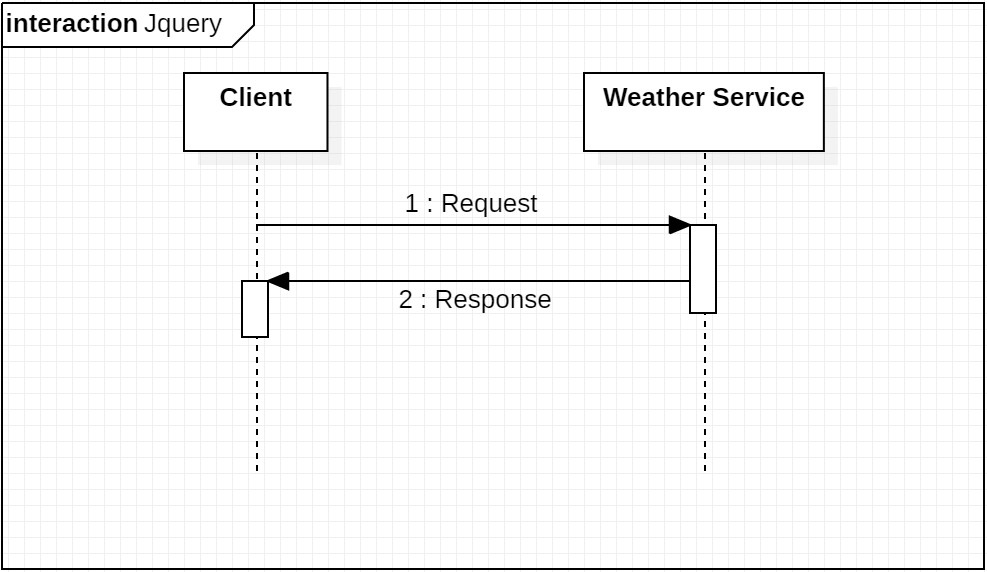


In the case of error, client will reconnect and prompt to check common user input mistakes.

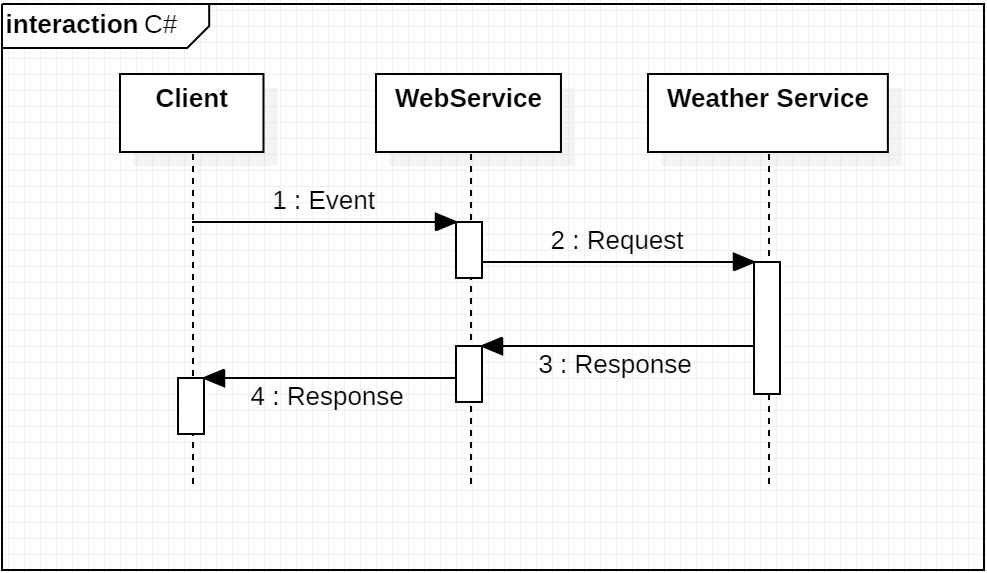


Sequence Diagrams

Jquery:



aspx:

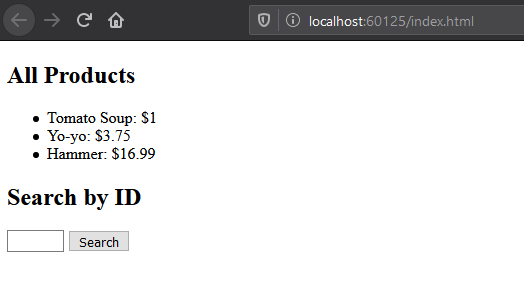


Task 2

Browser

Debug run the application and you will see the following page on load.

(There is a simulated loading, loading circle will appear)



Enter a number between 1 to 3 (ID for the products).



Postman

Localhost:

http://localhost:60125/

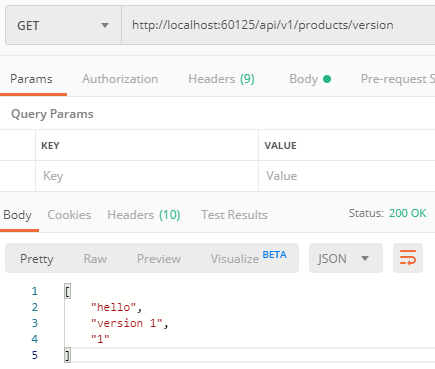
APIs:

|  |  |  |
| --- | --- | --- |
| Version | Type | API Route |
| V1 | GET | api/v1/products/version |
| V1 | GET | api/v1/products/message?name1=<**name**>&name2=<**name**>&name3=<**name**> |
| V1 | GET | api/v1/products |
| V1 | GET | api/v1/products/{**id**} |
| V2 | GET | api/v2/products |
| V2 | GET | api/v2/products/{**id**} |
| V2 | GET | api/v2/products?Category=<**category**> |
| V2 | POST | api/v2/products |
| V2 | PUT | api/v2/products/{**id**} |
| V2 | DELETE | api/v2/products/{**id**} |
| V3 | GET | api/v3/products |
| V3 | GET | api/v3/products/{**id**} |
| V3 | GET | api/v3/products?Category=<**category**> |
| V3 | POST | api/v3/products |
| V3 | PUT | api/v3/products/{**id**} |
| V3 | DELETE | api/v3/products/{**id**} |

GET: api/v1/products/version

Link: <http://localhost:25157/api/v1/products/version>

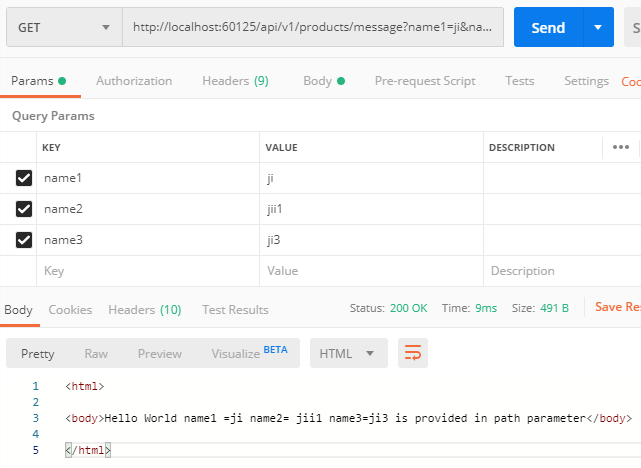
Get API version.



GET: api/v1/products/message?name1=<**name**>&name2=<**name**>&name3=<**name**>

Link: <http://localhost:60125/api/v1/products/message/name1=ji&name2=jii1&name3=ji3>

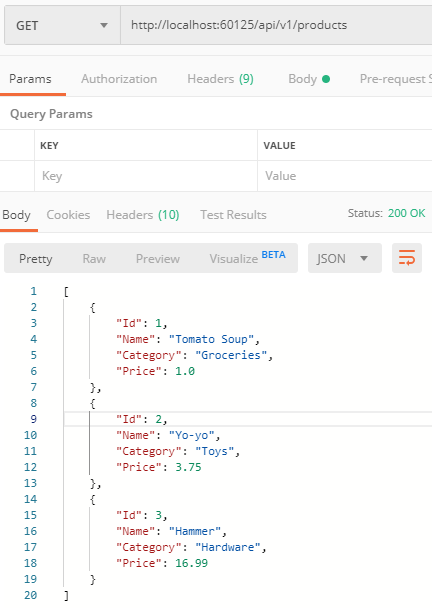
Get method with user defined parameters.



GET: api/v1/products

Link: <http://localhost:60125/api/v1/products>

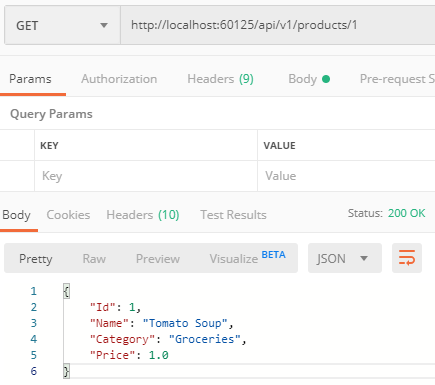
Get all products.



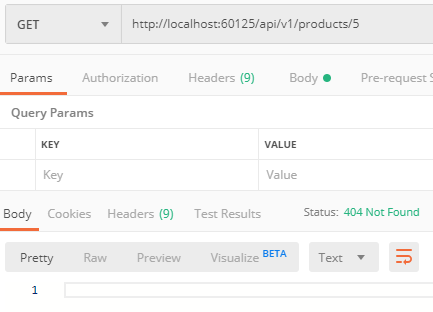
GET: api/v1/products/{**id**}

Link: <http://localhost:60125/api/v1/products/1>

Get product based on id. (Available Id between 1 to 3)



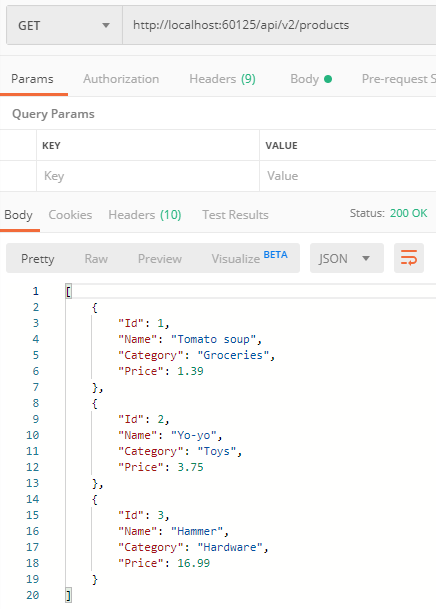
Providing invalid id will lead to 404 Not Found.



GET: api/v2/products

Link: <http://localhost:60125/api/v2/products>

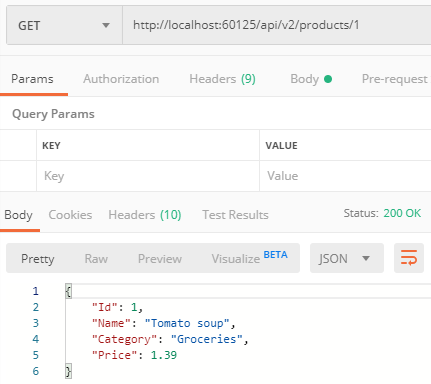
Get all products.



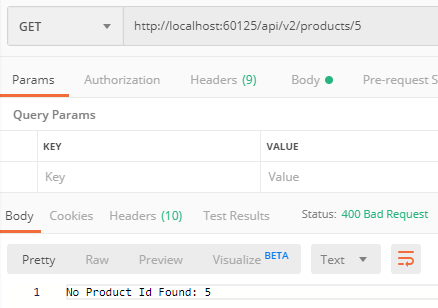
GET: api/v2/products/{**id**}

Link: <http://localhost:60125/api/V2/products>

Get product based on id. (Available Id between 1 to 3)



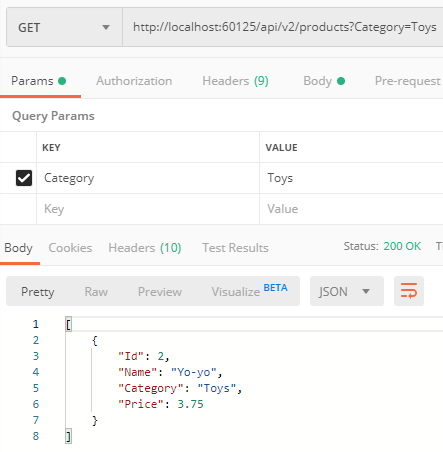
Providing invalid id will lead to 400 Bad Request



GET: api/v2/products?Category=<**category**>

Link: <http://localhost:60125/api/v2/products?Category=Toys>

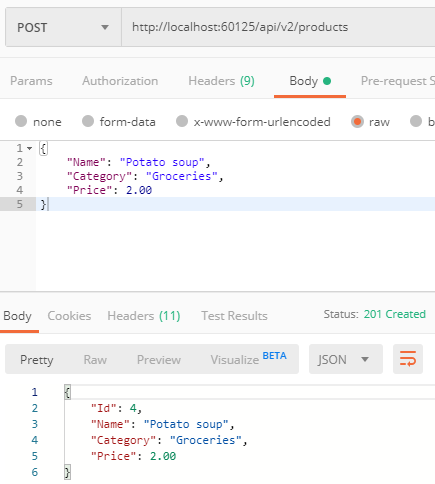
Get products by Category as parameter.



POST: api/v2/products

Link: <http://localhost:60125/api/v2/products>

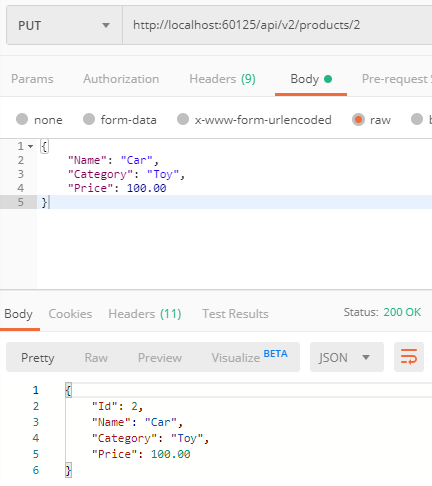
Create product using Post.



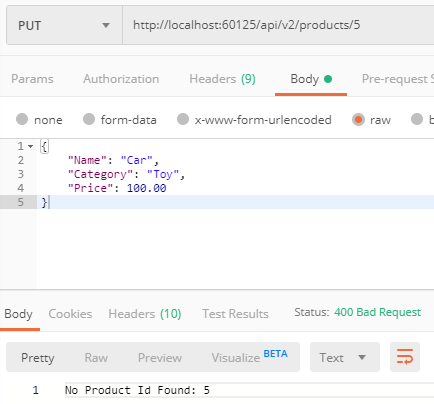
PUT: api/v2/products/{**id**}

Link: <http://localhost:60125/api/v2/products/2>

Change product details based on id using Put. (Available Id between 1 to 3)



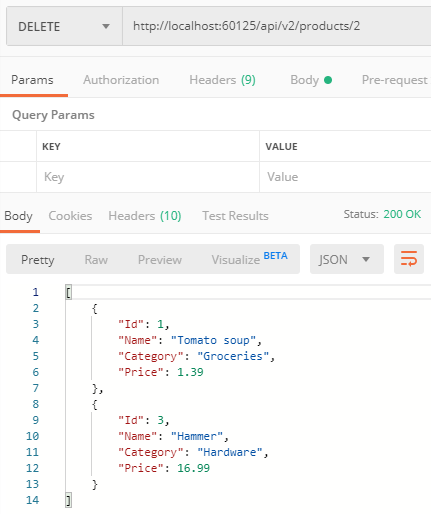
Providing invalid id will lead to 400 Bad Request



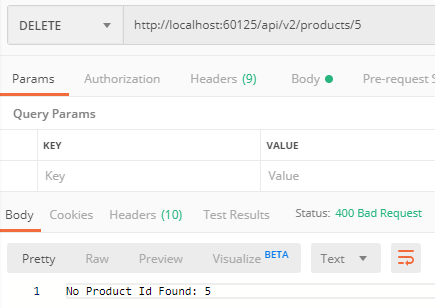
DELETE: api/v2/products/{**id**}

Link: <http://localhost:60125/api/v2/products/2>

Remove product based on id using Delete. (Available Id between 1 to 3)



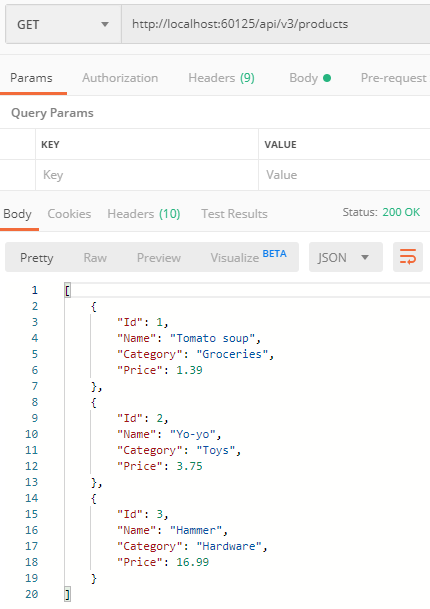
Providing invalid id will lead to 400 Bad Request



GET: api/v3/products

Link: <http://localhost:60125/api/v3/products>

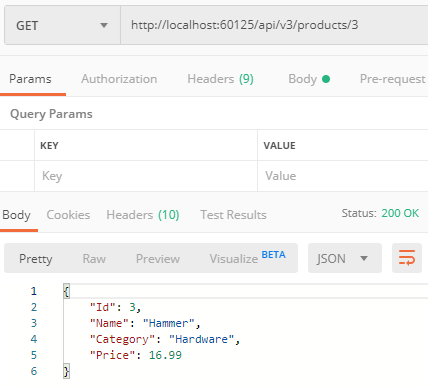
Get all products.



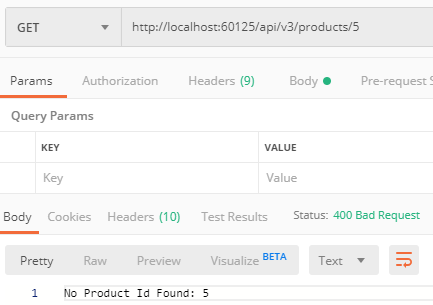
GET: api/v3/products/{**id**}

Link: <http://localhost:60125/api/v3/products/3>

Get product based on id. (Available Id between 1 to 3)



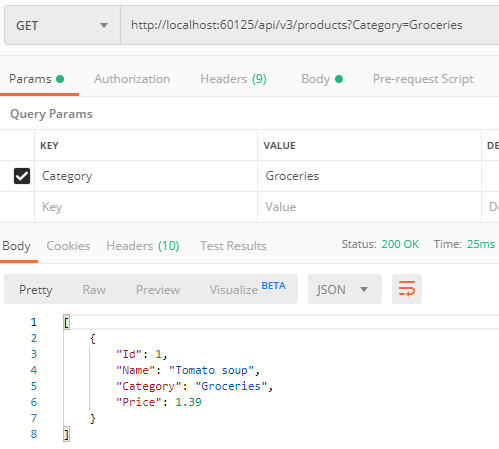
Providing invalid id will lead to 400 Bad Request



GET: api/v3/products?Category=<**category**>

Link: <http://localhost:60125/api/v3/products?Category=Groceries>

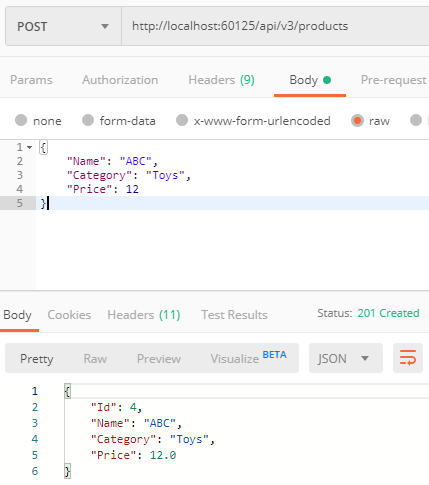
Get products by Category as parameter.



POST: api/v3/products

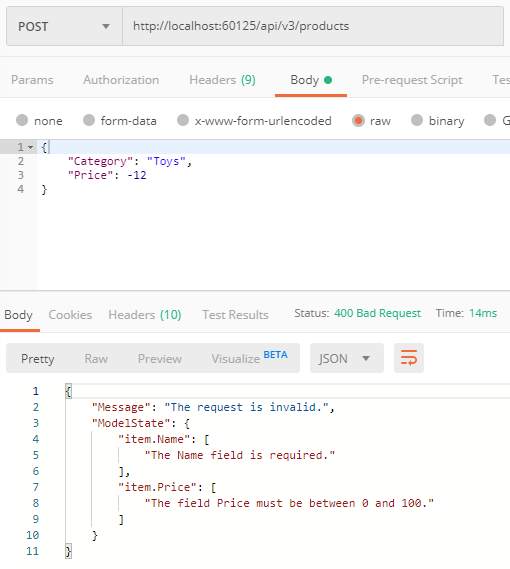
Link: <http://localhost:60125/api/v3/products>

Create product using Post.

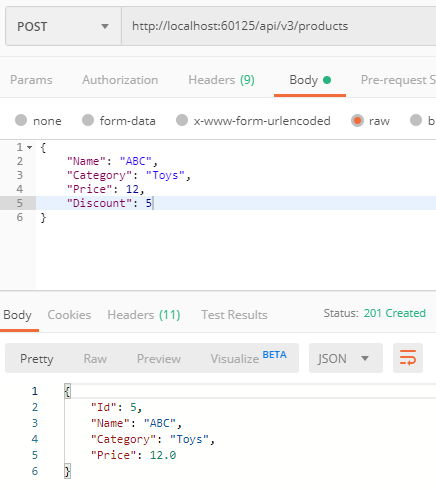


“Under-posting” will lead to 400 Bad Request. (Required: Name, Price\*)

*\*Price must be between 0 to 100*



“Over-posting” will lead to 201 Created. (“Over-post” values are excluded in creation)

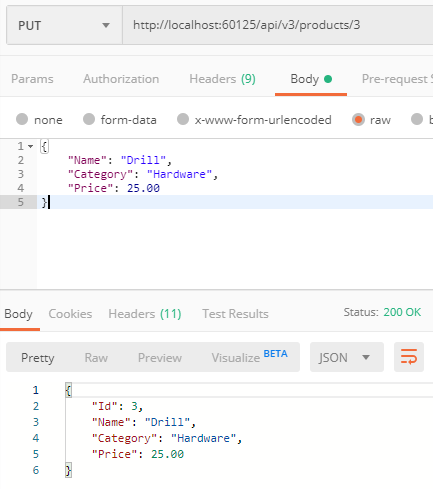


^ ”Discount” is excluded during creation

PUT: api/v3/products/{**id**}

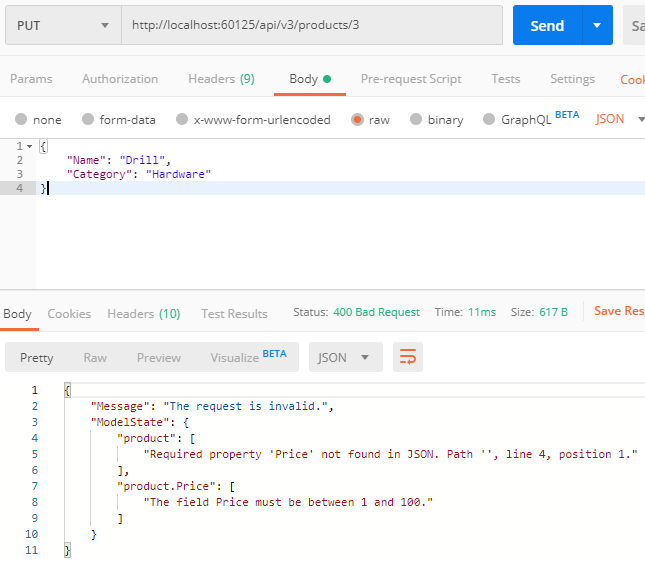
Link: <http://localhost:60125/api/v3/products/3>

Change product details based on id using Put. (Available Id between 1 to 3)

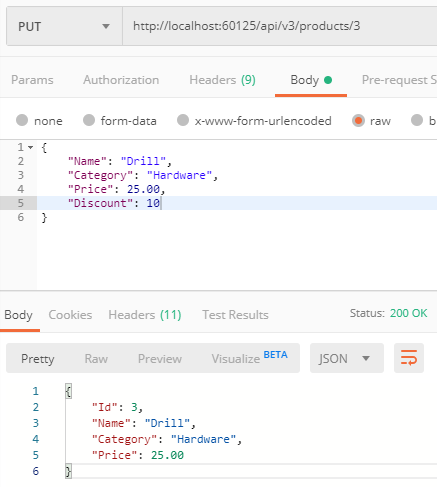


“Under-posting” will lead to 400 Bad Request. (Required: Name, Price\*)

*\*Price must be between 0 to 100*

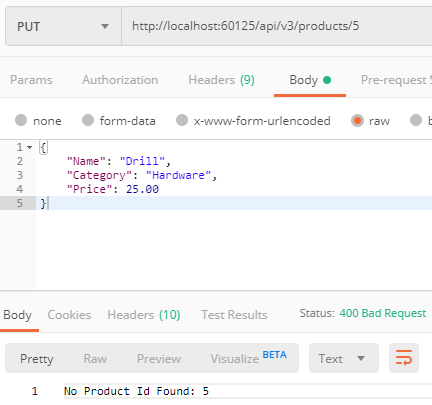


“Over-posting” will lead to 200 OK. (“Over-post” values are excluded in creation)



^ ”Discount” is excluded during creation

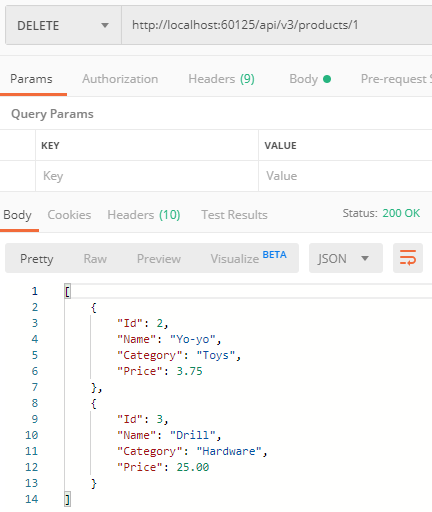
Providing invalid id will lead to 400 Bad Request



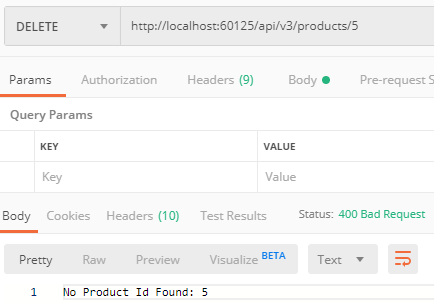
DELETE: api/v3/products/{**id**}

Link: <http://localhost:60125/api/v1/products/1>

Remove product based on id using Delete. (Available Id between 1 to 3)



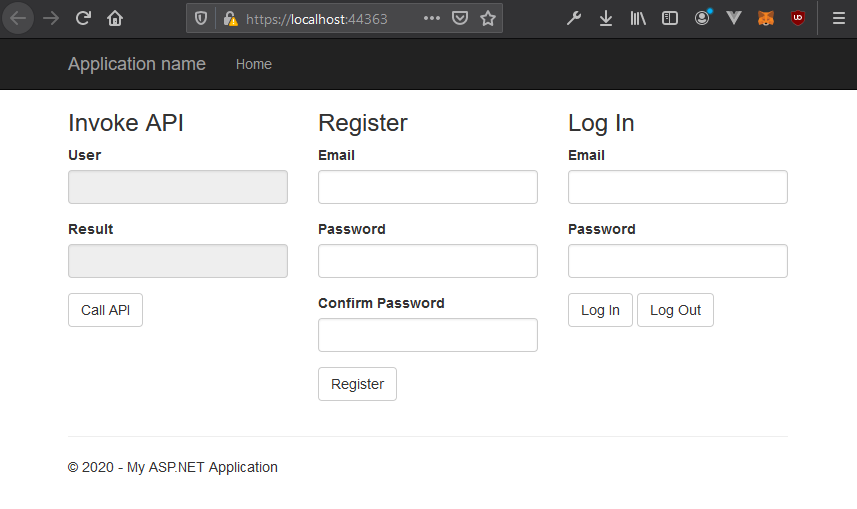
Providing invalid id will lead to 400 Bad Request



Task 3

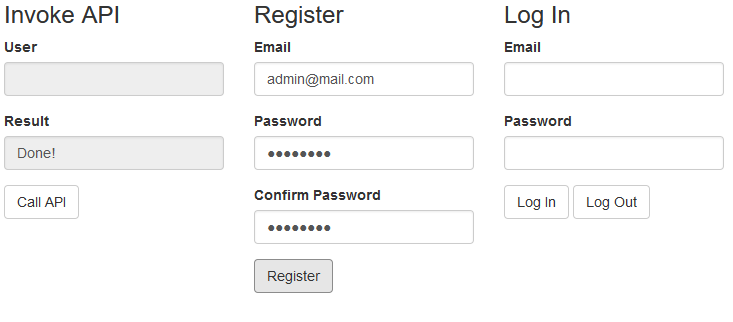
Browser

Debug run the application and you will see the following page on load.

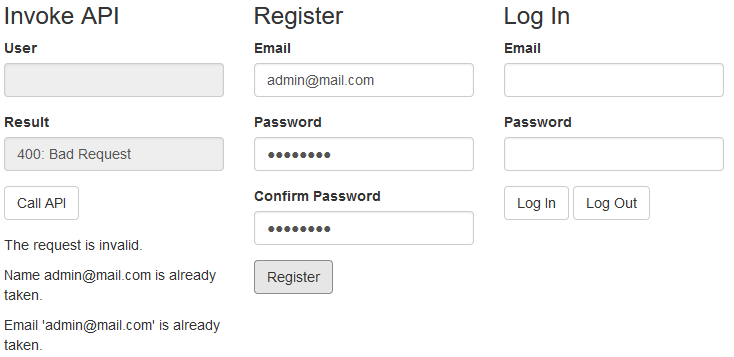


Register

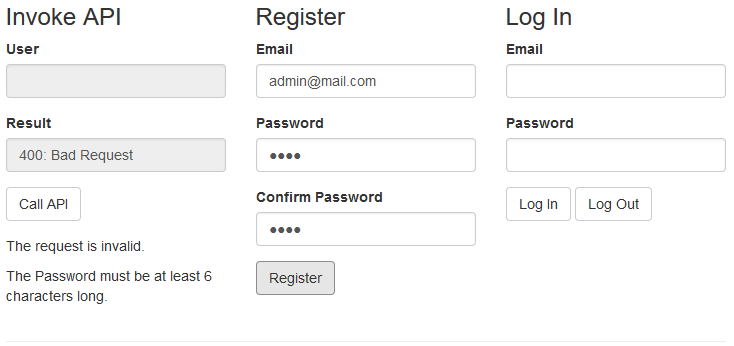
Register account by entering email and password twice.

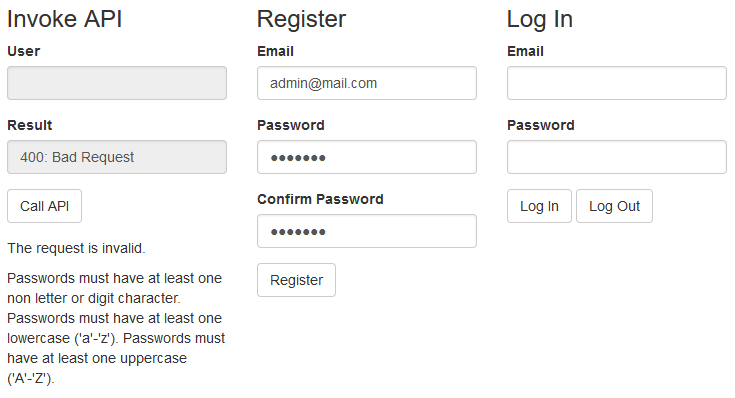


Register can fail if email is already existed in the database.



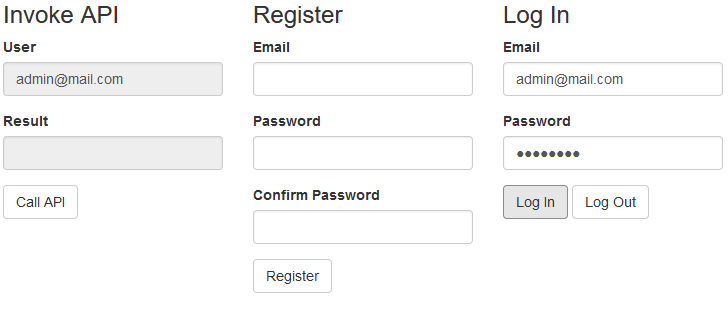
Register can fail if password fails the validation process.



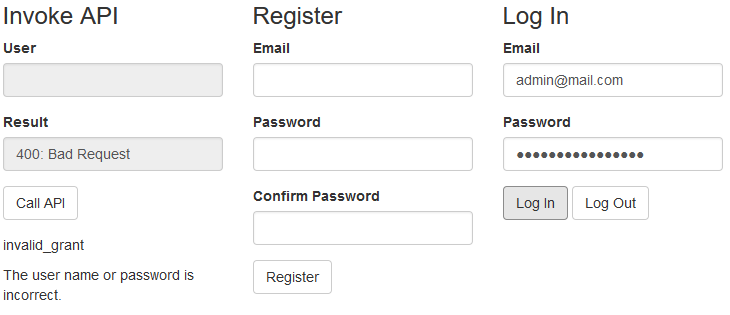


Log in

Log in to the account by entering email and password. (Bearer token will be created)

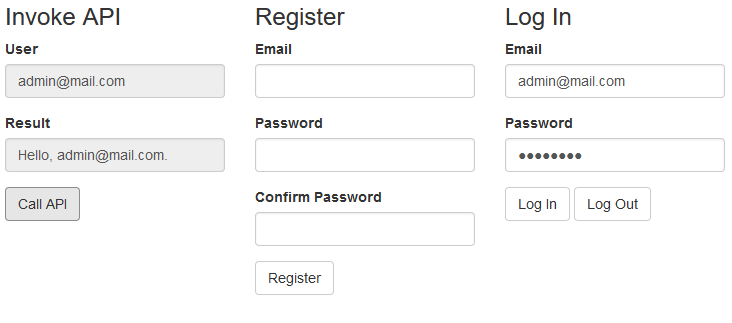


Log in can fail if email does not exist or password is incorrect.

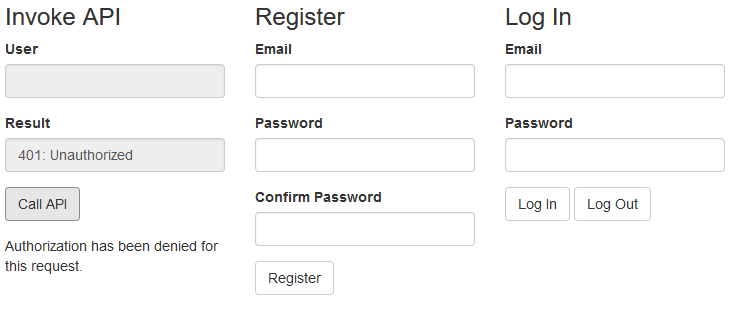


Call API

After logging in, user will be able to call API. (Authorized using bearer token)



API could not be called when user is not log on. (Due to lack of bearer token)



Postman

Localhost:

https://localhost:44363/

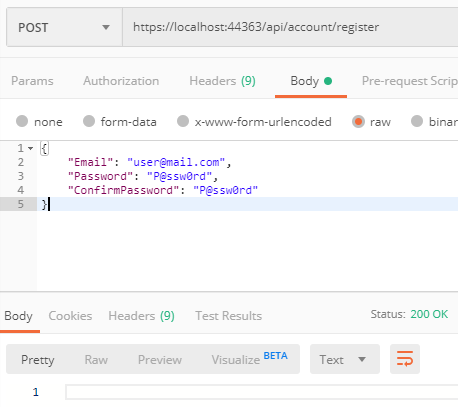
APIs:

|  |  |  |
| --- | --- | --- |
| Security Level | Type | API Route |
| Anonymous | POST | api/account/register |
| Anonymous | POST | token |
| Authorized | GET | api/values |

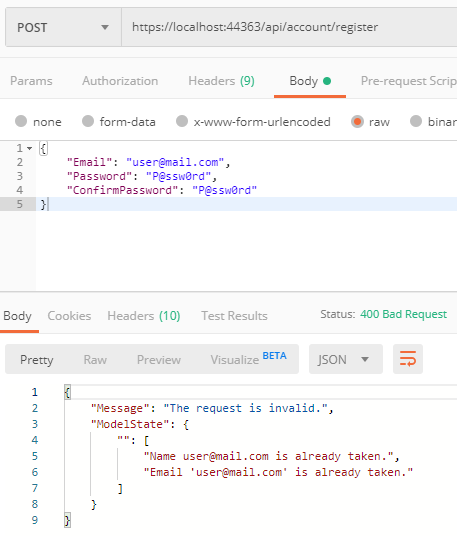
POST: api/account/register

Link: <https://localhost:44363/api/account/register>

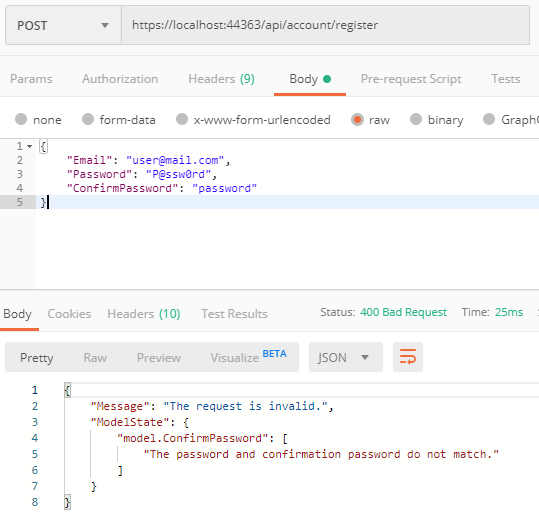
Register account by Post.



Exist email will lead to 400 Bad Request.



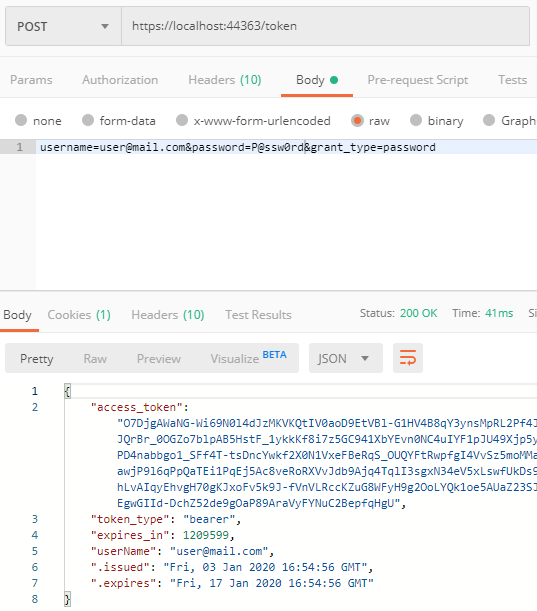
Fail model validation will lead to 400 Bad Request.



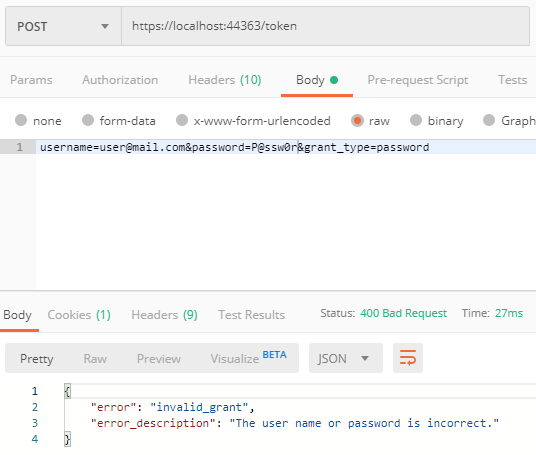
POST: token

Link: <https://localhost:44363/token>

Get token for API which need authorization by posting email and password. (Email is set as a ClaimType in the token)



Email that does not exist or incorrect password will lead to 400 Bad Request.

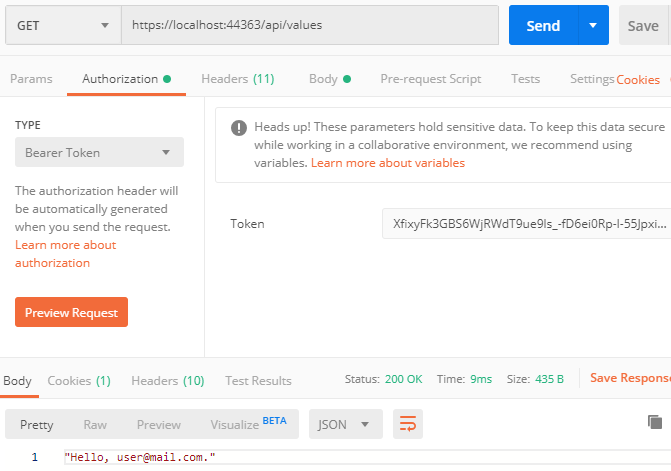


GET: api/values

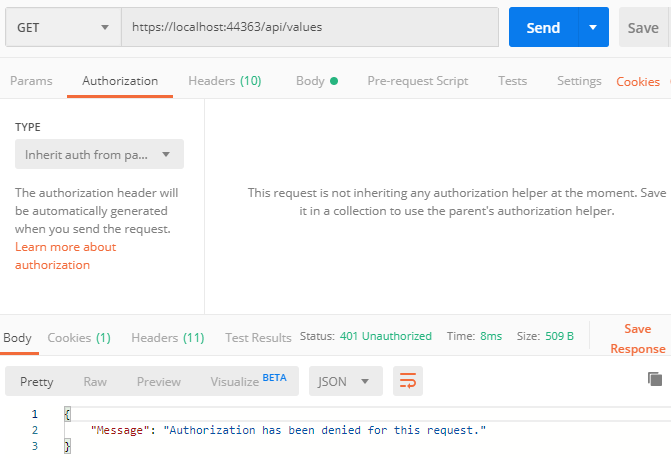
Link: <https://localhost:44363/api/values>

(Requires Bearer token)

Get request with bearer token. (Email is derived from a ClaimType in the token)



Get request without bearer token will lead to 403 Unauthorized .

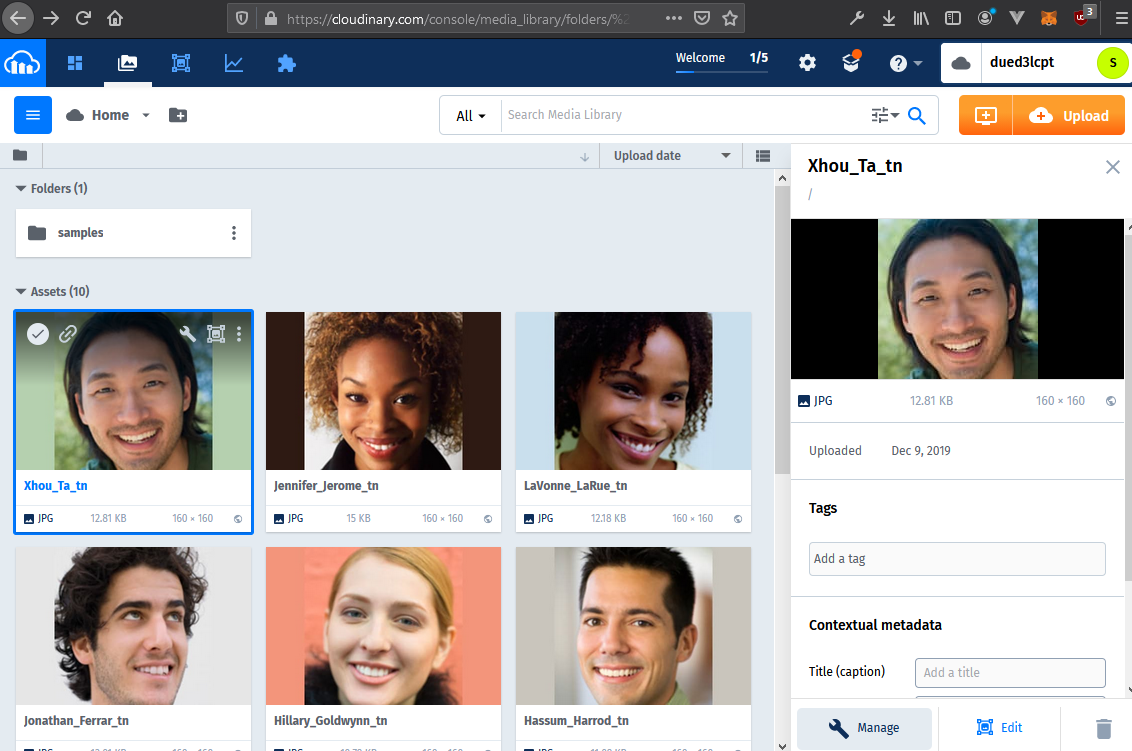


Task 4

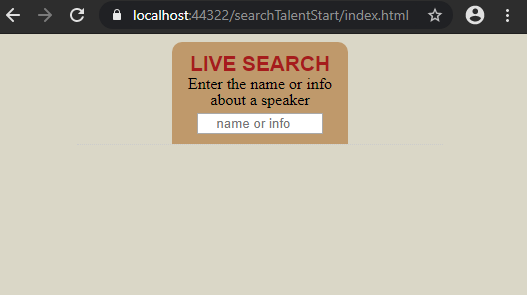
Browser

Assign your link to the Cloudinary media library <https://cloudinary.com/> .

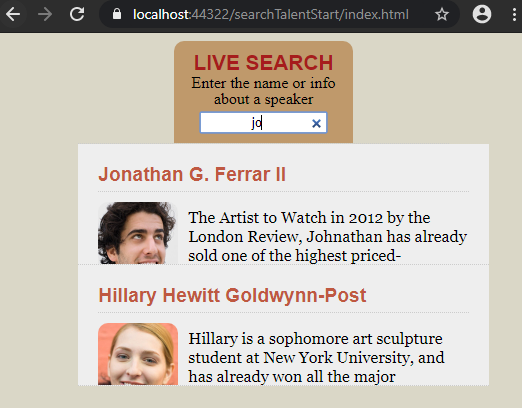




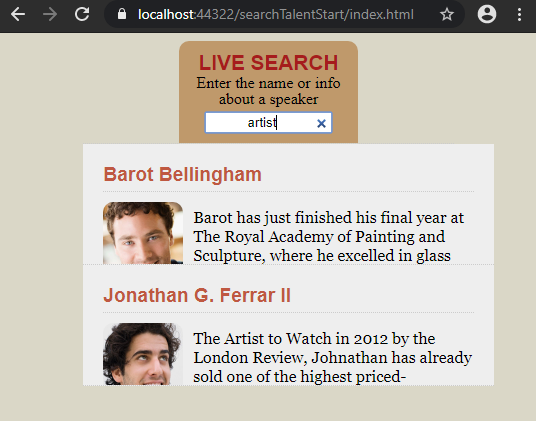
Debug run the application and you will see the following page on load.



You can search by typing the name.



You can also search by typing the info.



Postman

Localhost:

<https://localhost:44322/>

Cloud Host:

<https://productstoreweihan.azurewebsites.net>

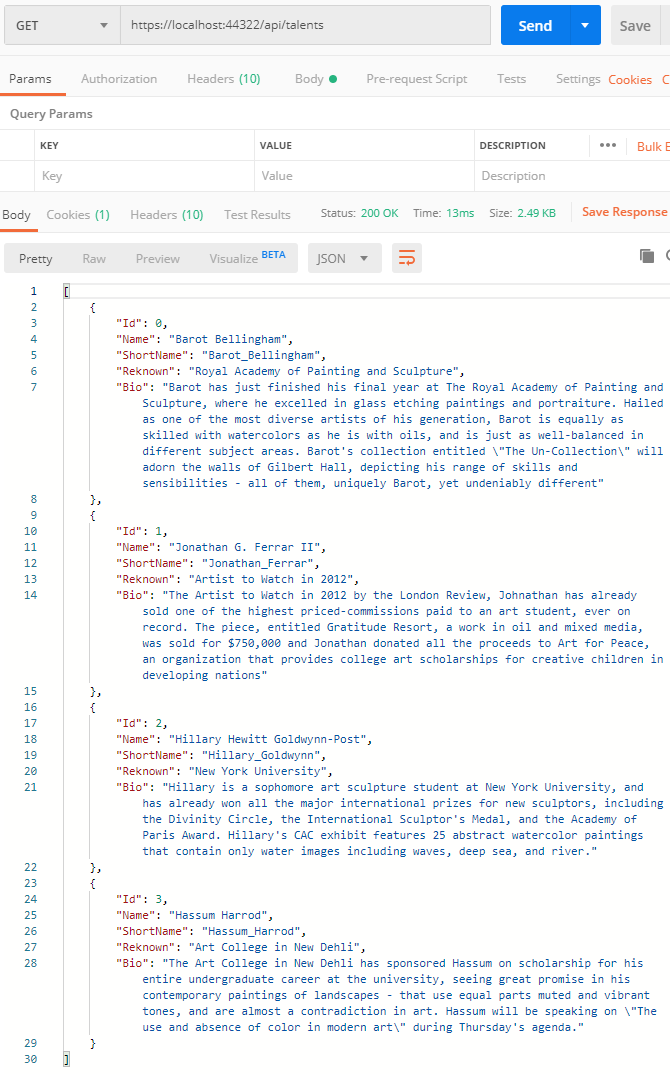
APIs:

|  |  |
| --- | --- |
| Type | API Route |
| GET | api/talents |
| GET | api/talents/{**id**} |

GET: api/talents

Link: <https://localhost:44322/api/talents>

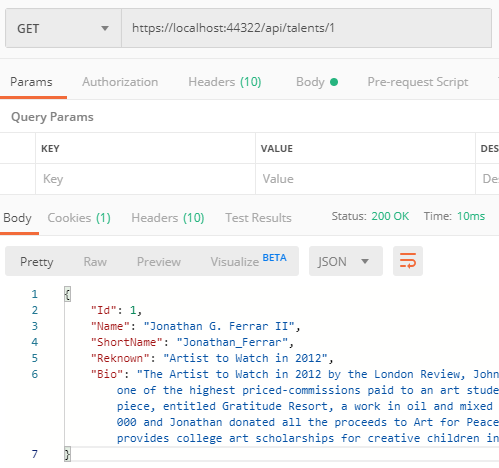
Get all talents



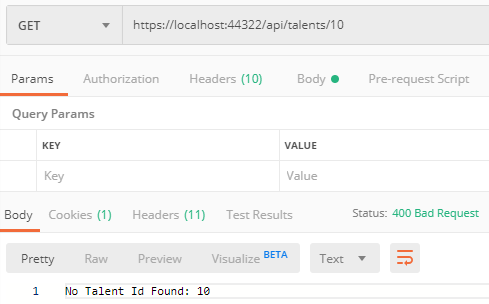
GET: api/talents/{**id**}

Link: <https://localhost:44322/api/talents/1>

Get talent based on Id. (Available Id between 0 to 3)



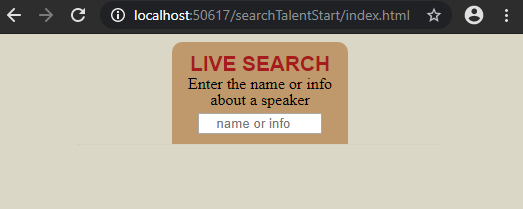
Providing invalid id will lead to 400 Bad Request



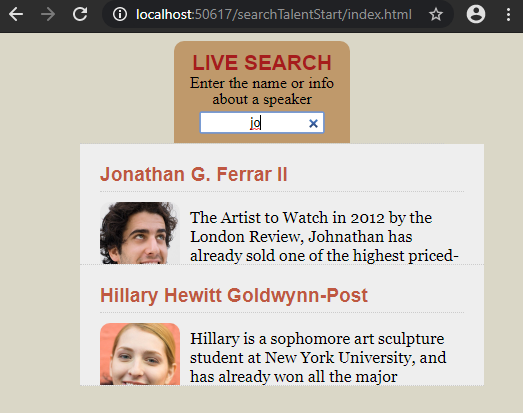
Task 5

Browser

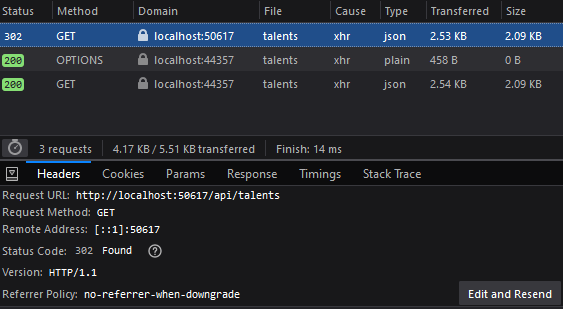
Debug run the application and you will see the following page on load.

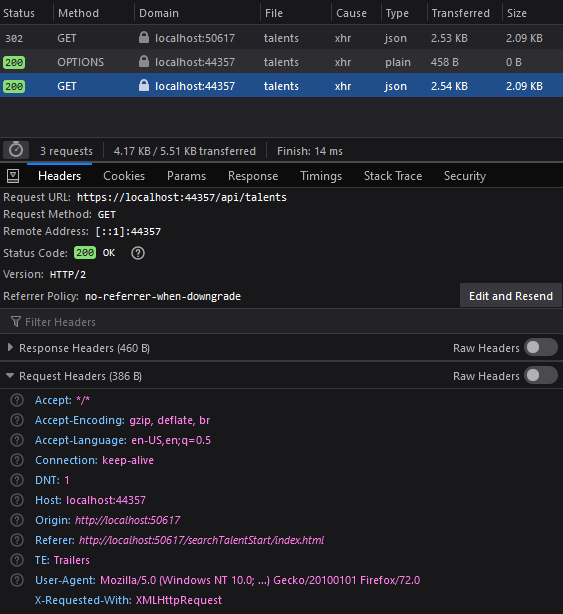


You can search by typing the name.



Using the browser console, we can see that https is used for data in transit.





Fiddler (Postman)

Localhost:

<http://localhost:50617/>

<https://localhost:44357/>

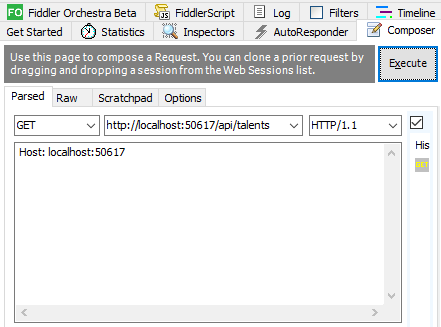
APIs:

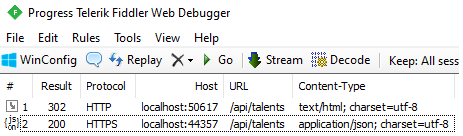
|  |  |
| --- | --- |
| Type | API Route |
| GET | api/talents |
| GET | api/talents/{**id**} |

GET: api/talents

Link: <http://localhost:50617/api/talents>

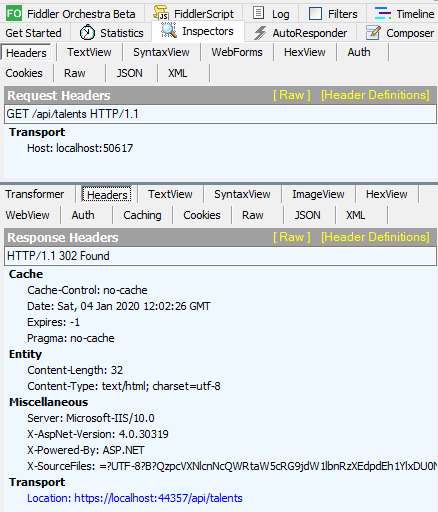
Get all talents using http will result a redirect to https

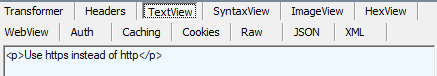




1. Header shows transport to <https://localhost:44357> with response 302 Found.

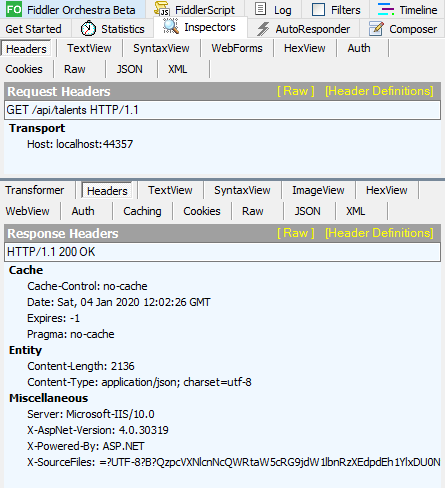






1. JSON can be view using <https://localhost:44357> with response 200 OK.



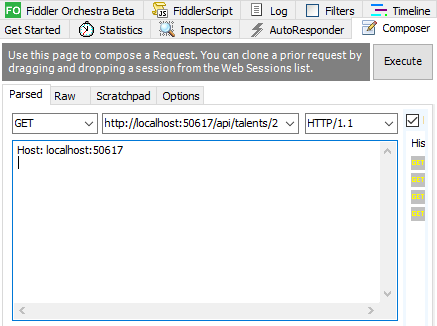


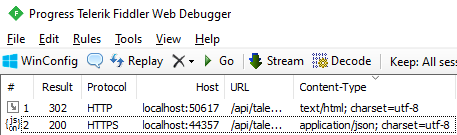


GET: api/talents/{**id**}

Link: <http://localhost:50617/api/talents/2>

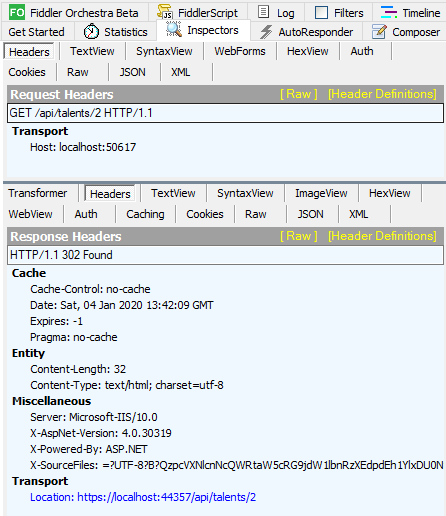
Get talent using http will result a redirect to https. (Available Id between 0 to 3)

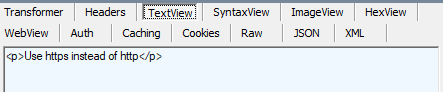




1. Header shows transport to <https://localhost:44357> with response 302 Found.

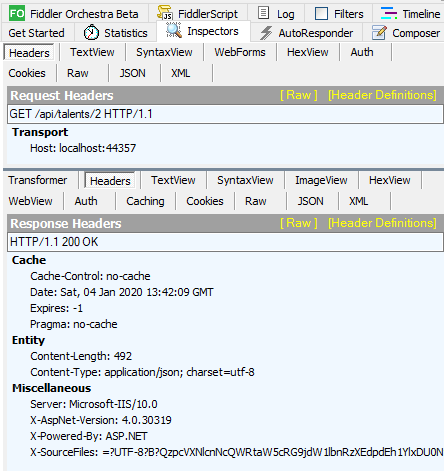


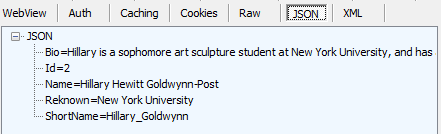




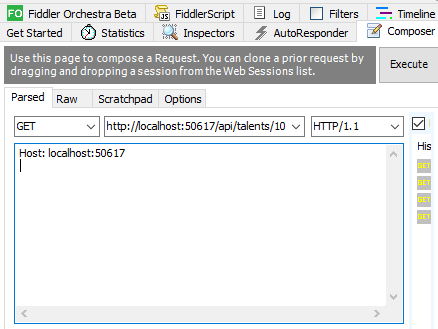
1. JSON can be view using <https://localhost:44357> with response 200 OK.

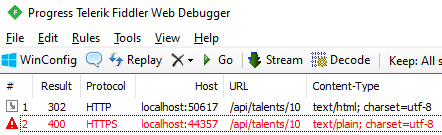






Entering invalid Talent id will still redirect to https but with 400 Bad Request.

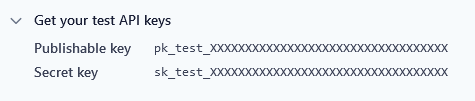


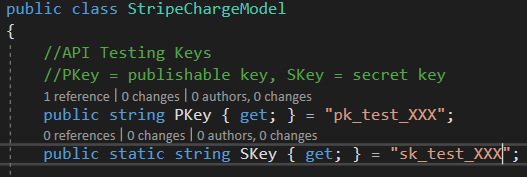


Task 6

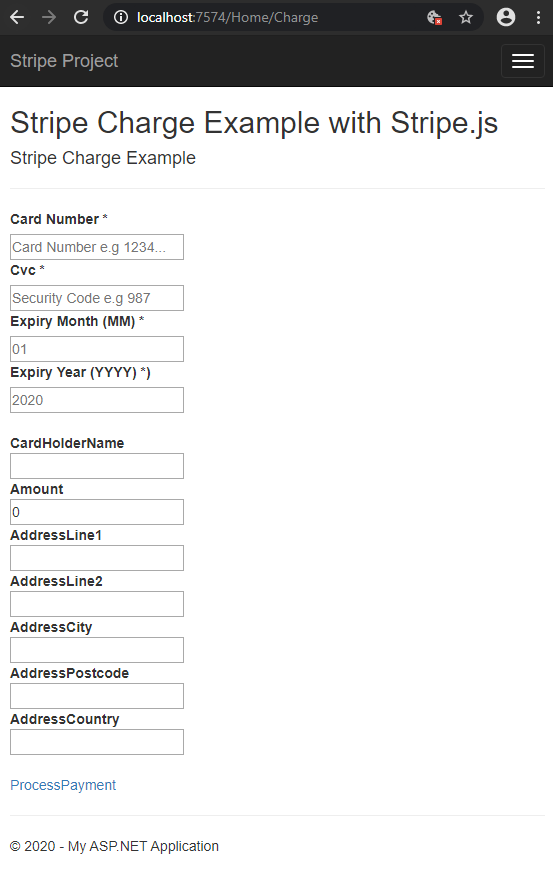
Browser

Insert your publishable key and secret key inside StripeChargeModel.cs from <https://dashboard.stripe.com/test/dashboard> .





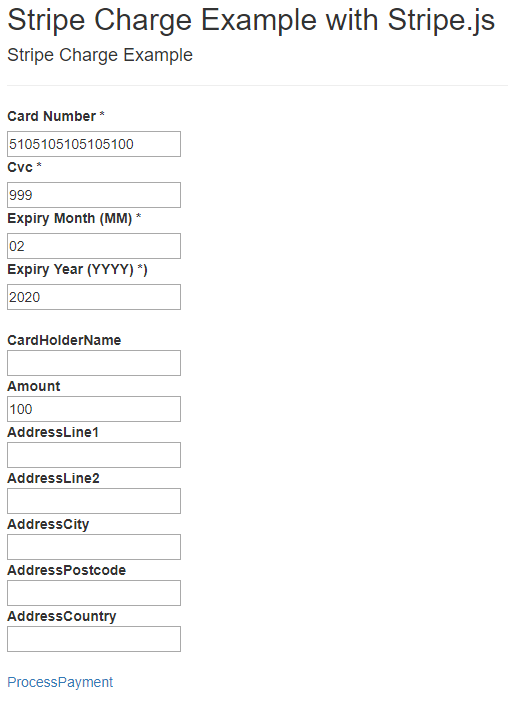
Debug run the application and you will see the following page on load.



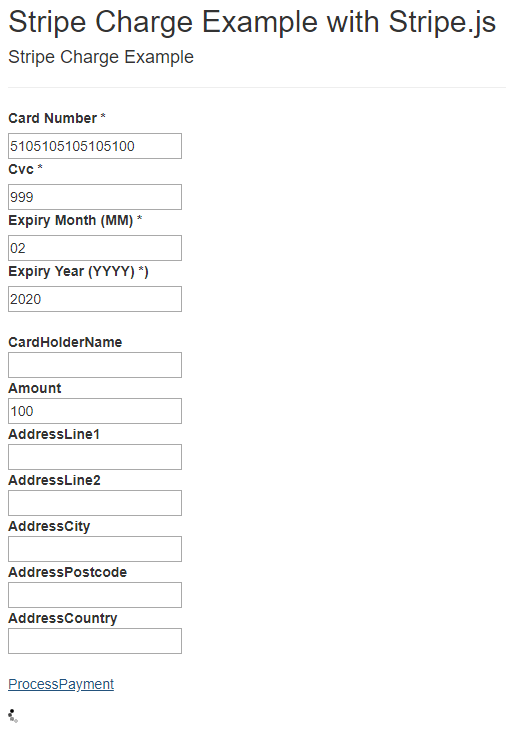
Fill in the details

(Card Holder Name and Addresses are optional)

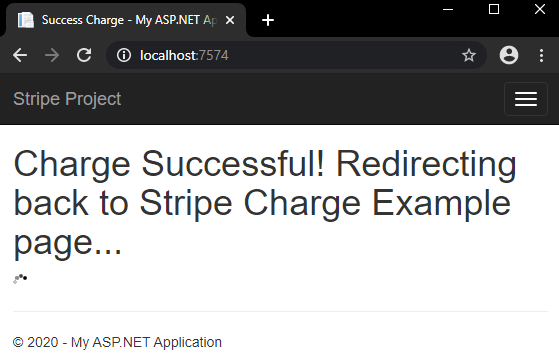
(Minimum amount: 0.01)



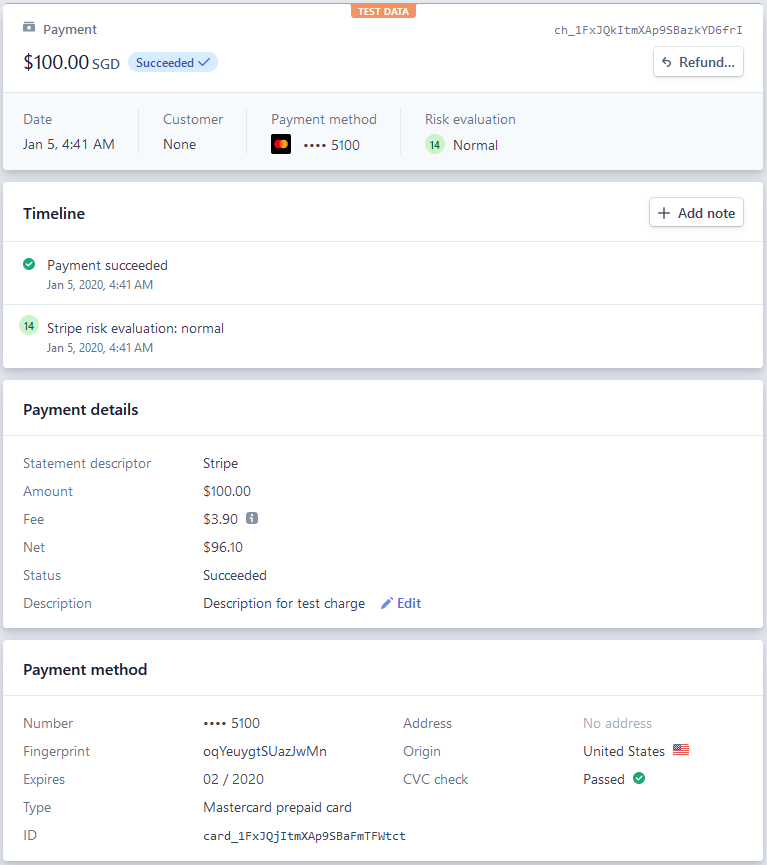
Stripe will start validating card, create token and charge.  
(indicated by the loading icon)



Successful charge will result in successful page with redirection back.

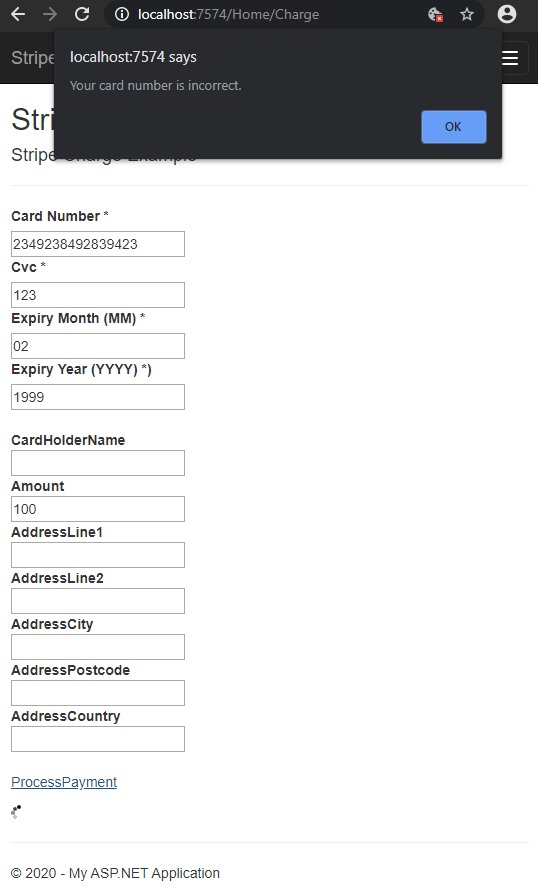


Successful charge will be reflected in <https://dashboard.stripe.com/test/payments> .

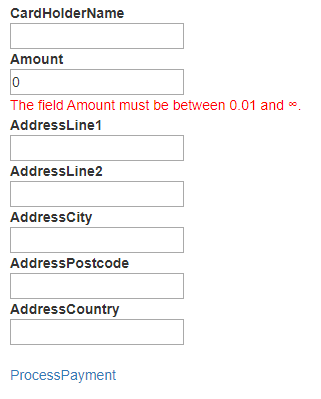


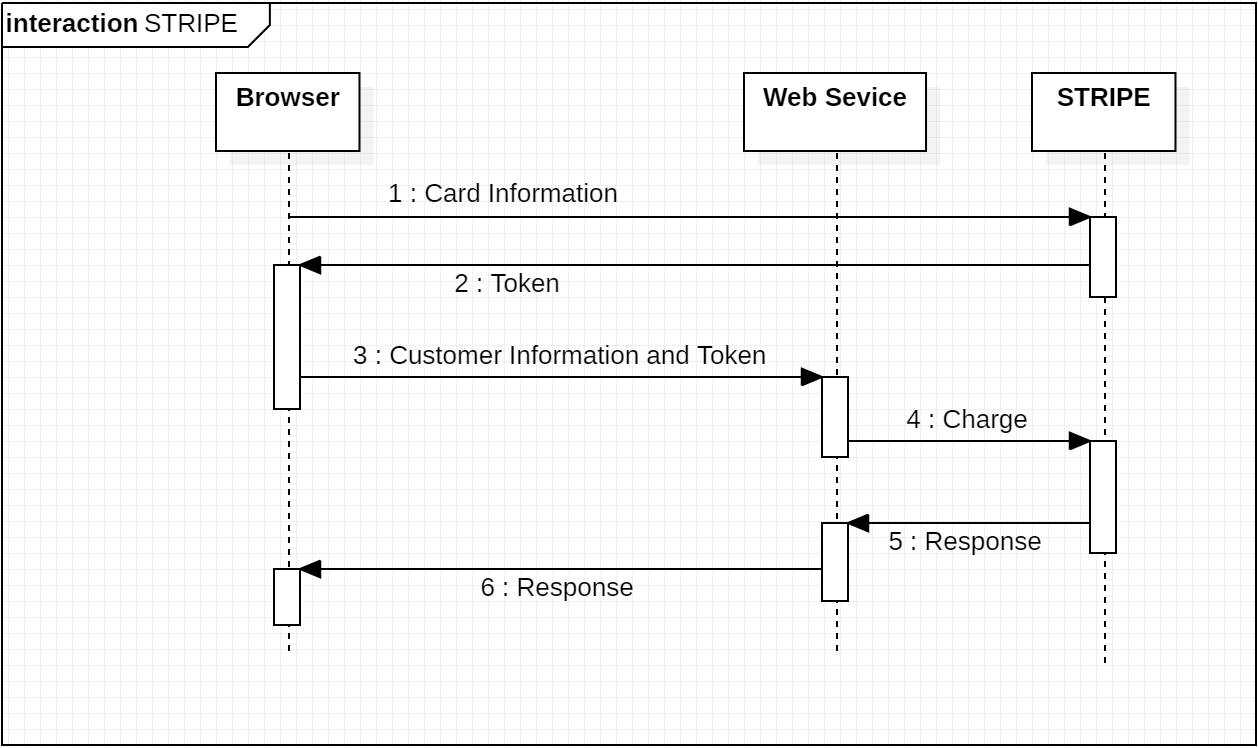


Error from Stripe if card details are invalid .

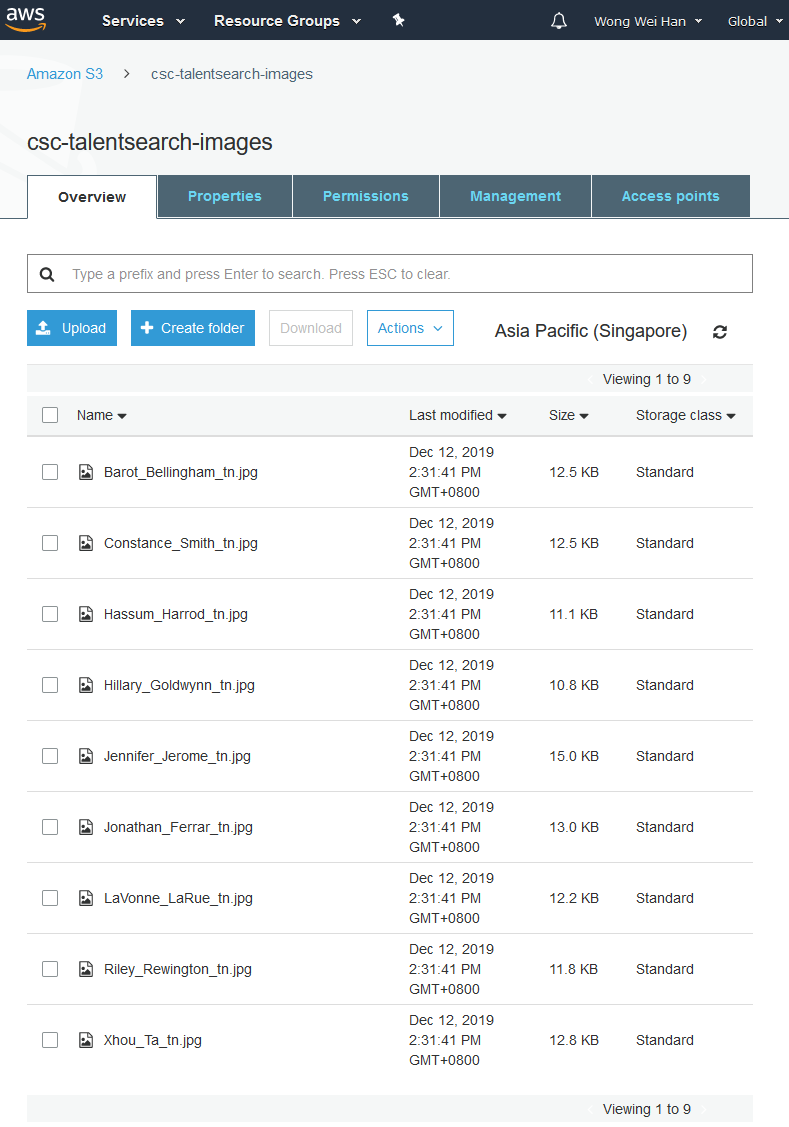


Error from Model Validation if amount is invalid.



Sequence Diagrams

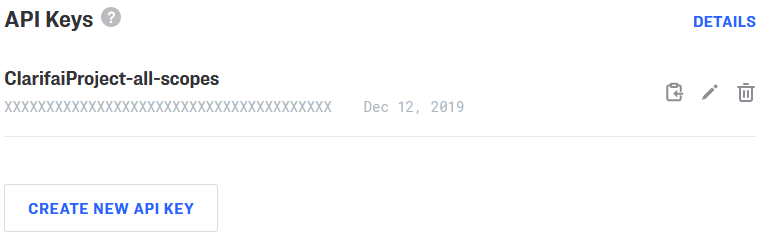
AWS Talents Images

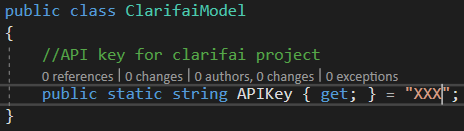


Task 7

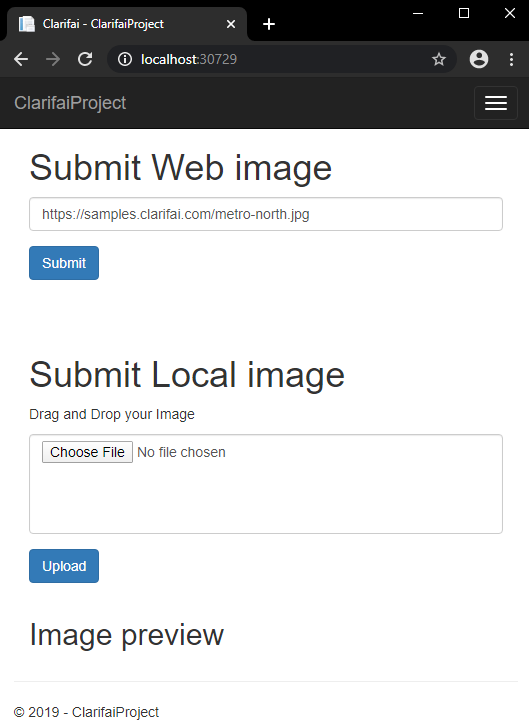
Browser

Insert your Clarifai API key inside ClarifaiModel.cs from <https://portal.clarifai.com/apps> .





Debug run the application and you will see the following page on load.



Submit: Web Image

Submit image using web image and wait for tag to load.

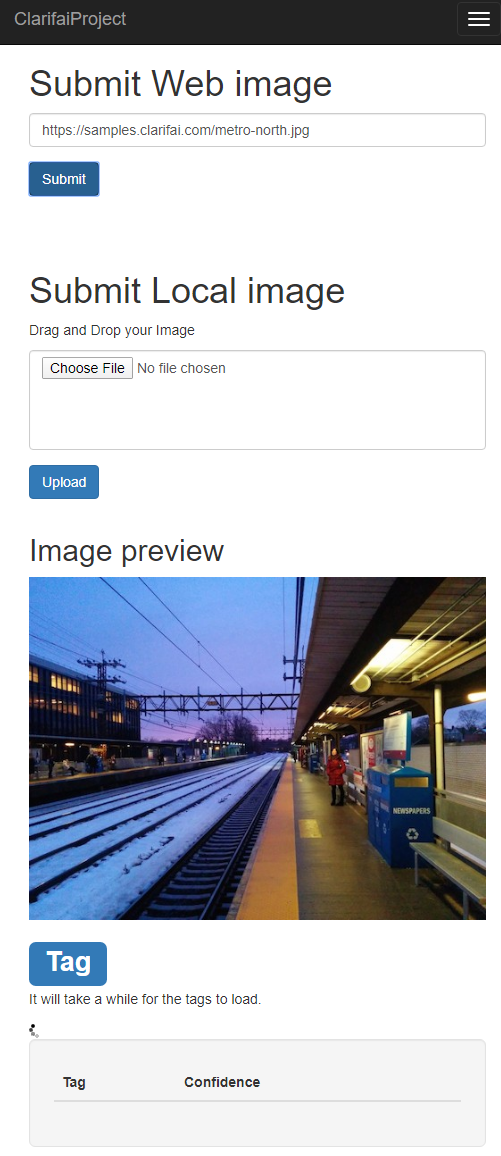
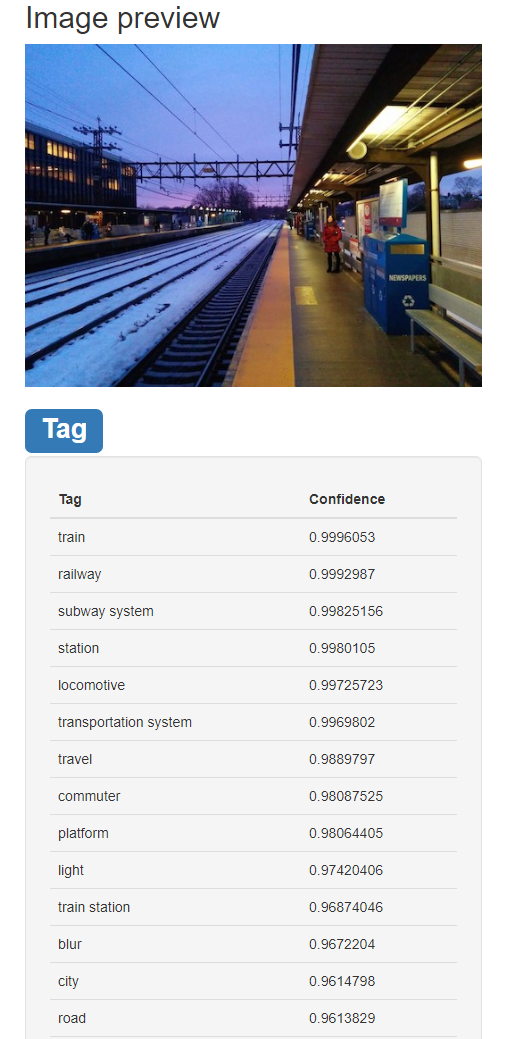
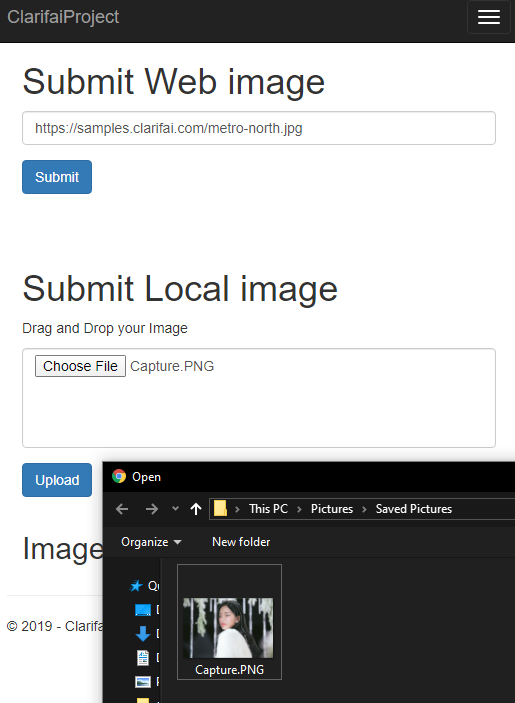


Image tag will be generated after a few seconds.



Submit: Local Image

Image can be uploaded from local computer via drag and drop.



Uploaded image will undergo tagging.

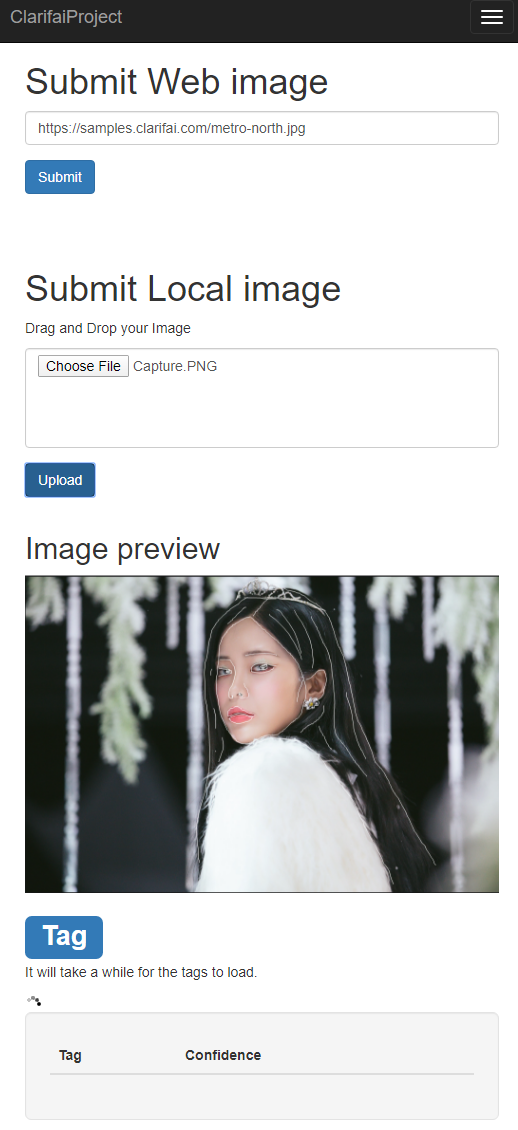


Image tag will be generated after a few seconds.

