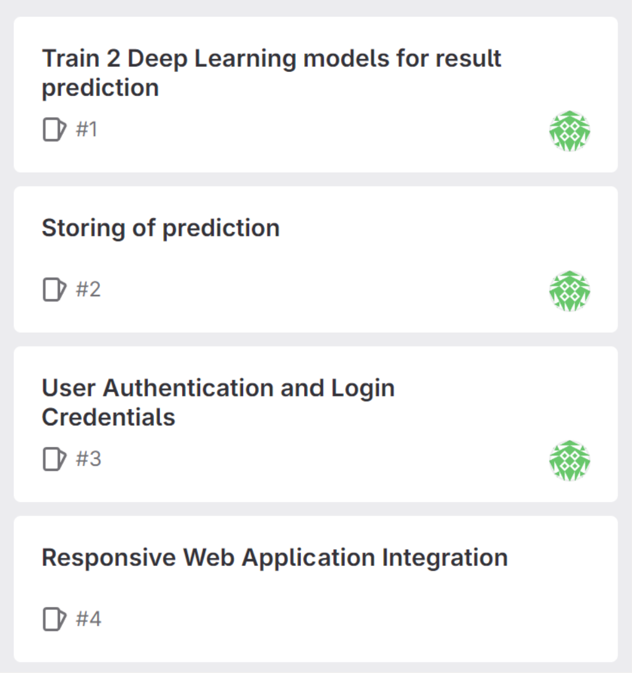
Vegetable Image Prediction

Done By: Toh Kien Yu (2222291)

Table Of Content

1. DevOps (Scrum Board and Branches)
2. Wireframes
3. Deep Learning Model Development
4. Web Application Development
5. Automatic Testing
6. MLOps CI/CD
7. Internet Deployment
8. Robotic Process Automation (RPA)

Scrum Board



Train 2 Deep Learning models for result prediction

#1

Storing of prediction

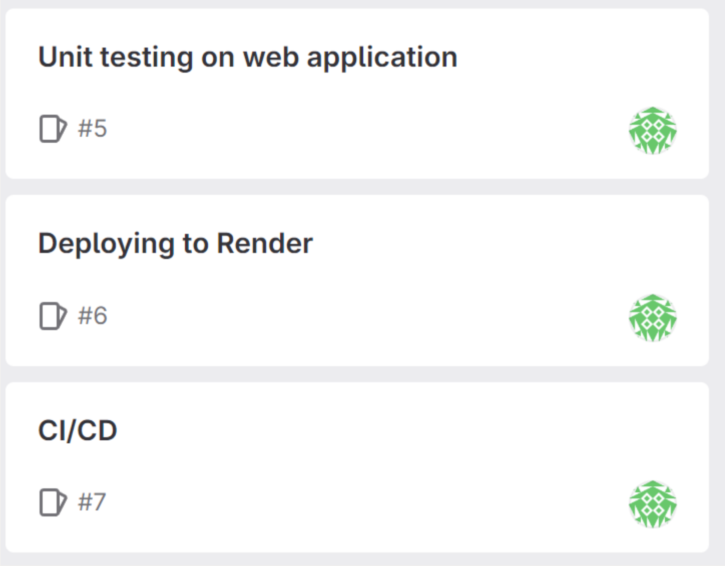
#2

User Authentication and Login Credentials

#3

Responsive Web Application Integration

#4



Unit testing on web application

#5

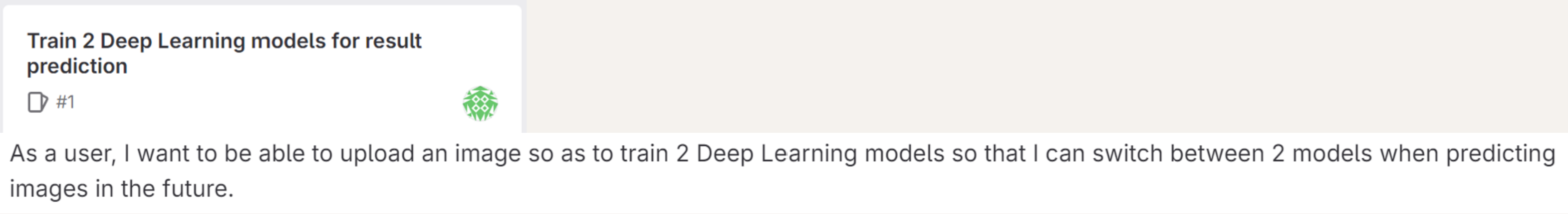
Deploying to Render

#6

CI/CD

#7

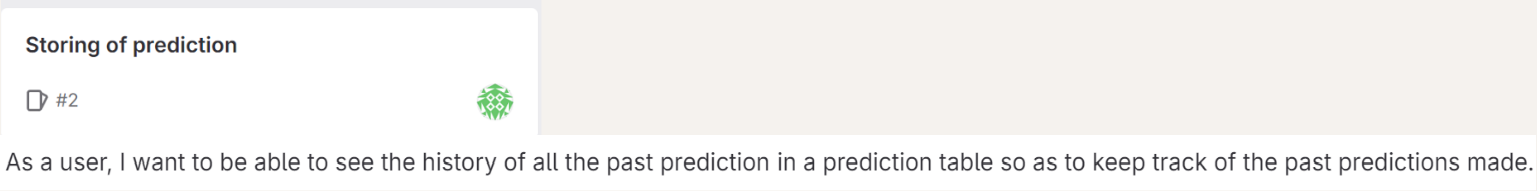
Scrum Board



Train 2 Deep Learning models for result prediction

#1

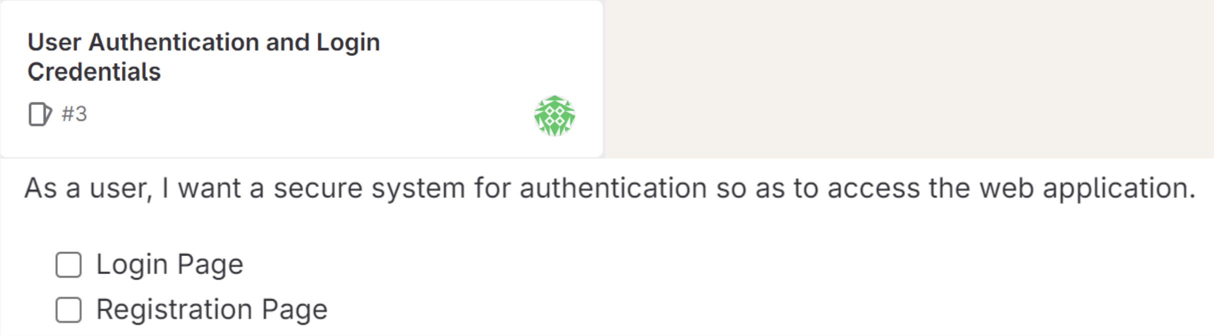
As a user, I want to be able to upload an image so as to train 2 Deep Learning models so that I can switch between 2 models when predicting images in the future.



Storing of prediction

#2

As a user, I want to be able to see the history of all the past prediction in a prediction table so as to keep track of the past predictions made.



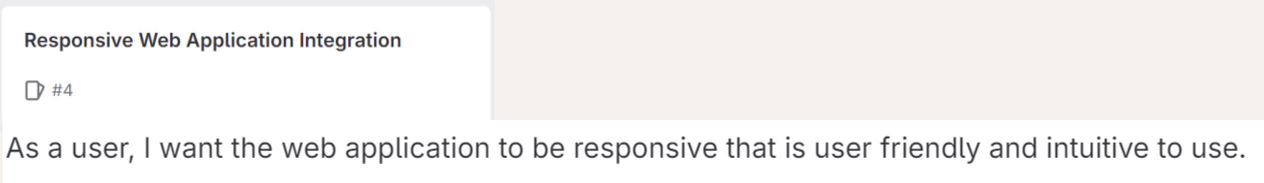
User Authentication and Login Credentials

#3

As a user, I want a secure system for authentication so as to access the web application.

- ☐ Login Page
- ☐ Registration Page

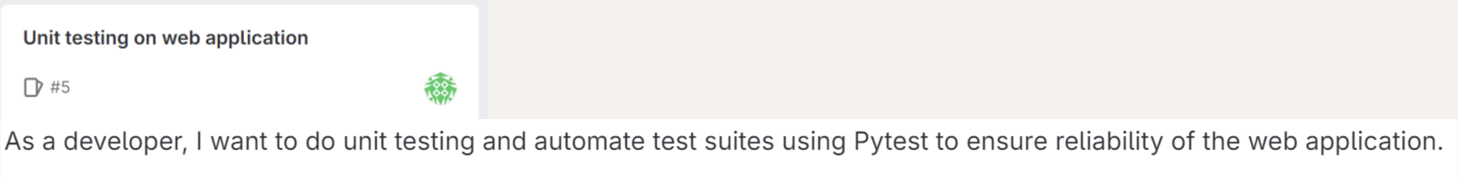
Scrum Board



Responsive Web Application Integration

#4

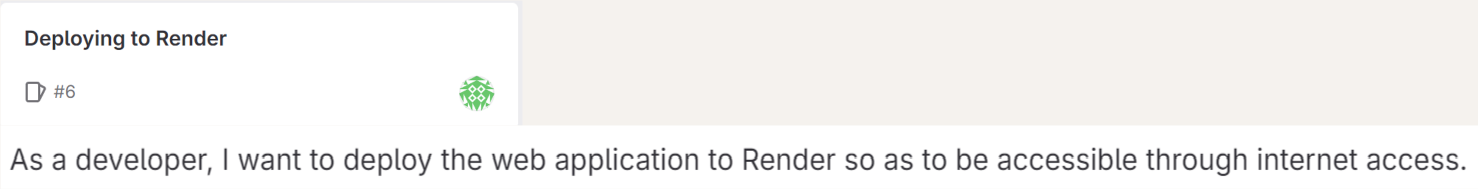
As a user, I want the web application to be responsive that is user friendly and intuitive to use.



Unit testing on web application

#5

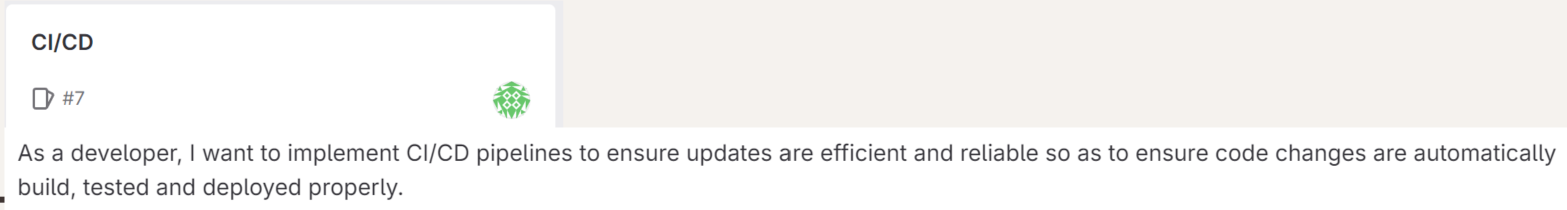
As a developer, I want to do unit testing and automate test suites using Pytest to ensure reliability of the web application.



Deploying to Render

#6

As a developer, I want to deploy the web application to Render so as to be accessible through internet access.

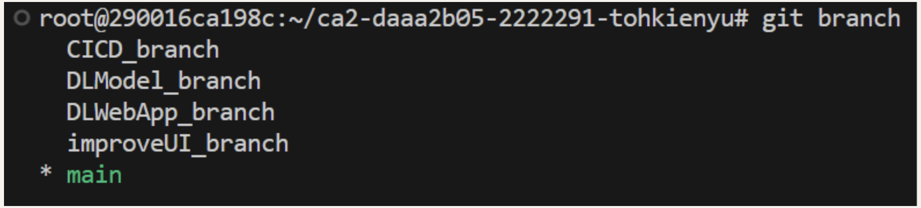


CI/CD

#7

As a developer, I want to implement CI/CD pipelines to ensure updates are efficient and reliable so as to ensure code changes are automatically build, tested and deployed properly.

Branches



```
○ root@290016ca198c:~/ca2-daaa2b05-2222291-tohkienyu# git branch
    CICD_branch
    DLModel_branch
    DLWebApp_branch
    improveUI_branch
  * main
```

Branch 1: DLModel_branch

- Focuses on the development and fine-tuning of Deep Learning models
- Containerized the 2 best models into a single Docker container
- Deploy Docker container to Render.com
- Unit testing to ensure models are correctly that both models can be switched for inference within the web application

Branch 2: DLWebApp_branch

- Integrate both models to web application for user to make predictions
- Designing of user interface such as (login, register, prediction and prediction history page)
- Unit testing

Branch 3: improveUI_branch

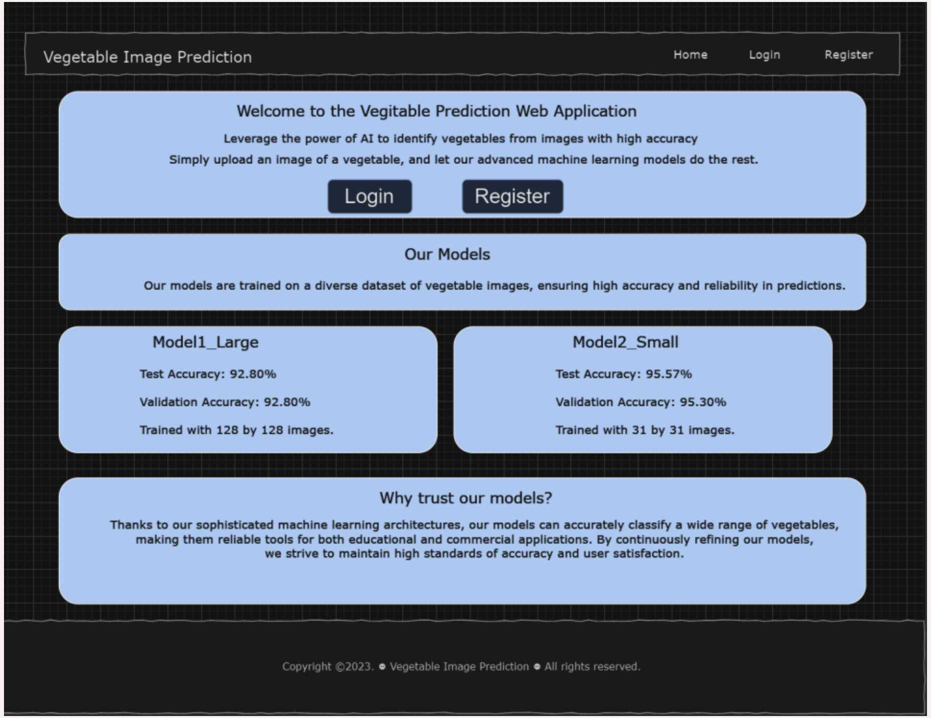
- Make website responsive
- Improve website user interface such as the layout, visuals and navigation.

Branch 4: CICD_branch

- Focuses on automating the Continuous Integration and Continuous Deployment process for the web application.
- Setting up of automated pipelines for testing and deployment ensuring efficient delivery of updates

WireFrame

index.html



Welcome to the Vegitable Prediction Web Application

Leverage the power of AI to identify vegetables from images with high accuracy

Simply upload an image of a vegetable, and let our advanced machine learning models do the rest.

Our Models

Our models are trained on a diverse dataset of vegetable images, ensuring high accuracy and reliability in predictions.

Model1_Large

Test Accuracy: 92.80%

Validation Accuracy: 92.80%

Trained with 128 by 128 images.

Model2_Small

Test Accuracy: 95.57%

Validation Accuracy: 95.30%

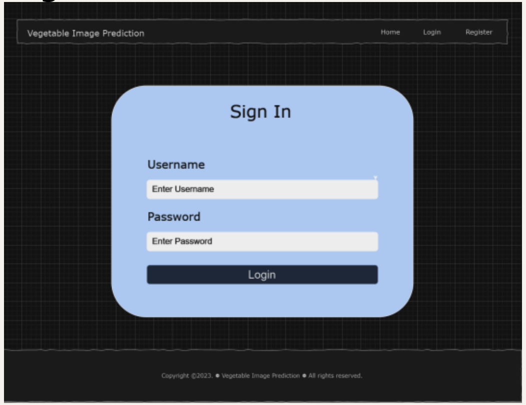
Trained with 31 by 31 images.

Why trust our models?

Thanks to our sophisticated machine learning architectures, our models can accurately classify a wide range of vegetables, making them reliable tools for both educational and commercial applications. By continuously refining our models, we strive to maintain high standards of accuracy and user satisfaction.

Copyright ©2023. ● Vegetable Image Prediction ● All rights reserved.

login.html



Sign In

Username

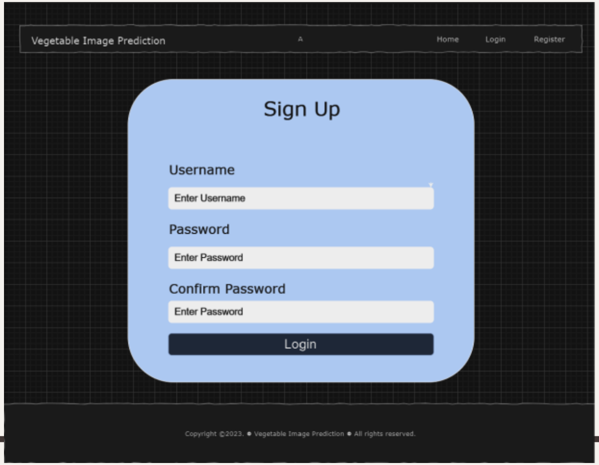
Enter Username

Password

Enter Password

Copyright ©2023. ● Vegetable Image Prediction ● All rights reserved.

register.html



Sign Up

Username

Enter Username

Password

Enter Password

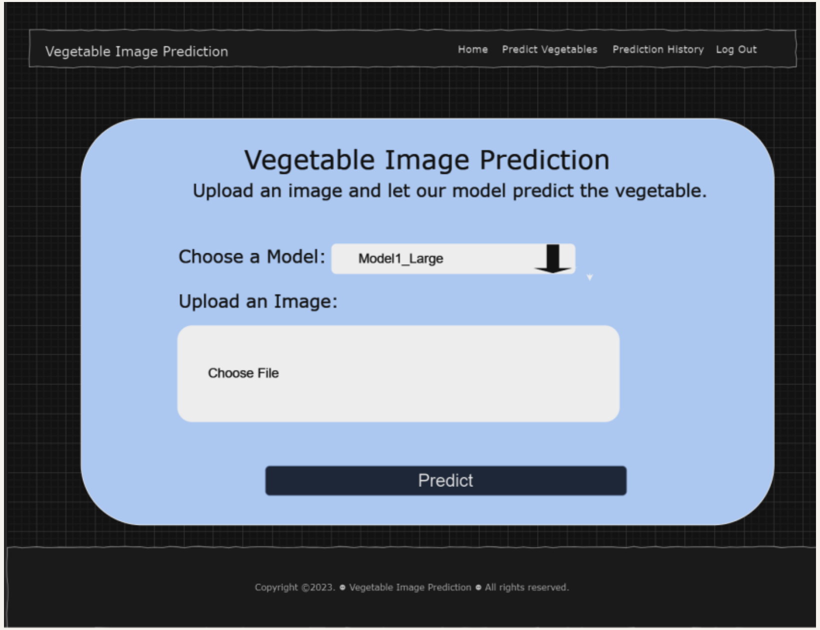
Confirm Password

Enter Password

Copyright ©2023. ● Vegetable Image Prediction ● All rights reserved.

WireFrame

predict.html



Vegetable Image Prediction

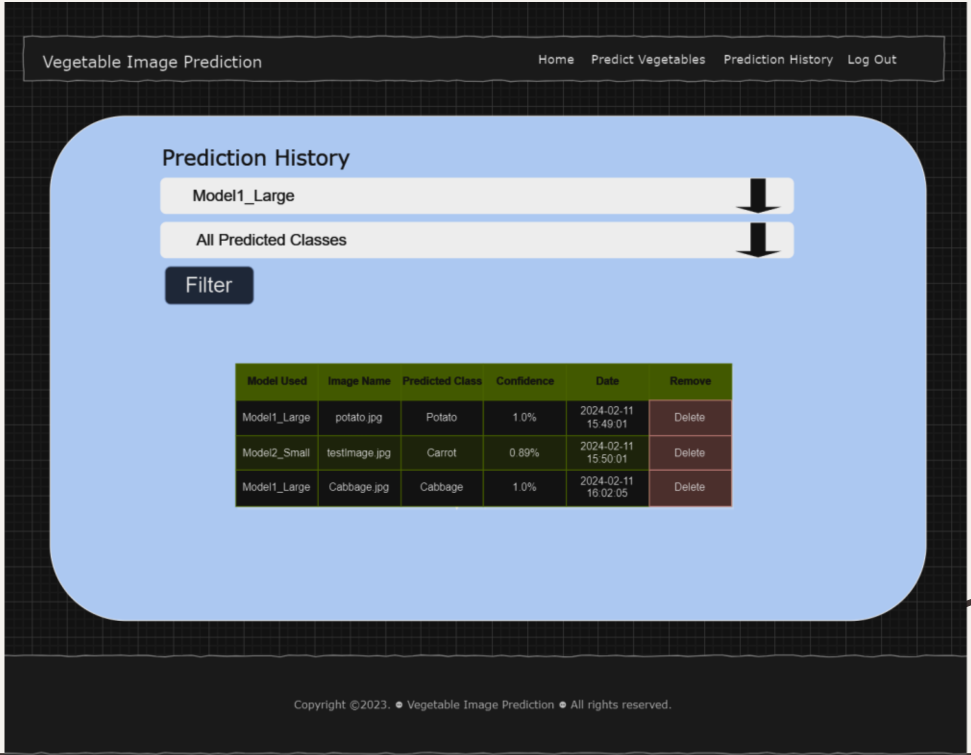
Upload an image and let our model predict the vegetable.

Choose a Model: Model1_Large

Upload an Image:

Copyright ©2023. • Vegetable Image Prediction • All rights reserved.

predictionHistory.html



Prediction History

Model1_Large

All Predicted Classes

Model Used	Image Name	Predicted Class	Confidence	Date	Remove
Model1_Large	potato.jpg	Potato	1.0%	2024-02-11 15:49:01	Delete
Model2_Small	testImage.jpg	Carrot	0.89%	2024-02-11 15:50:01	Delete
Model1_Large	Cabbage.jpg	Cabbage	1.0%	2024-02-11 16:02:05	Delete

Copyright ©2023. • Vegetable Image Prediction • All rights reserved.

Branch 1 (DLModel_branch)

DLModel_branch will focus on the development and optimization of Deep Learning models. We will then containerize the 2 models into a single Docker container before deploying to Render.com.

Upon Model development and after improvement this are the results for our best 2 models:

Final Model 1 (Trained on 31 by 31 images):
Test Accuracy: 95.57%
Validation Accuracy: 95.30%

Final Model 2 (Trained on 128 by 128 images):
Test Accuracy: 92.80%
Validation Accuracy: 92.80%

Deployment

1. Save models weights
2. Containerize 2 models into 1 container
3. Local deployment testing
4. Create Dockerfile
5. Deploy model on Render.com

Branch 1 (DLModel_branch)

Deployment

1. Save models weights

   First, I will first save the weights of the best performing 128x128 and 31x31 model.

2. Containerize 2 models into 1 container

   To containerize the 128x128 and 31x31 models into a single Docker container, I used the following command:

   ```
   docker run --name digit_server_CA2 -p 8501:8501 -v "C:/Users/kieny/Documents/Y2S2 - DAAA/DOAA CA2/ca2-daaa2b05-2222291-tohkienyu-main-img_classifier (1)/ca2-daaa2b05-2222291-tohkienyu-main-img_classifier/img_classifier:/models/img_classifier" -v "C:/Users/kieny/Documents/Y2S2 - DAAA/DOAA CA2/ca2-daaa2b05-2222291-tohkienyu-main-img_classifier (1)/ca2-daaa2b05-2222291-tohkienyu-main-img_classifier/img_classifier/models.config:/models/models.config" -t tensorflow/serving --model_config_file=/models/models.config
   ```

3. Local deployment testing

   After containerizing the 2 models, I deployed them locally and did unit testing to ensure it is functioning and verifying that the models were correctly loaded and able to make predictions with the expected level of accuracy.

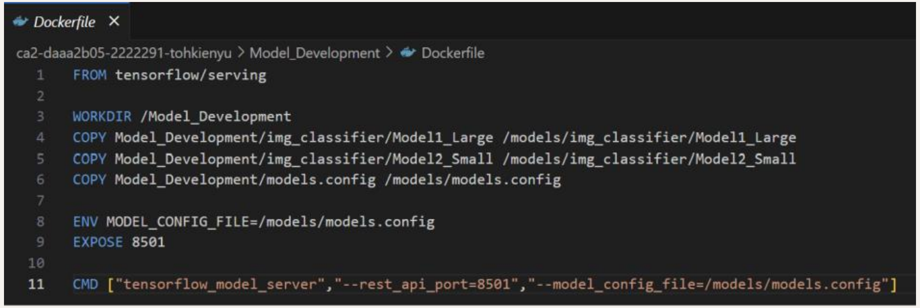
Branch 1 (DLModel_branch)

Deployment

4. Dockerfile For Deployment
   I then created a Dockerfile which contains the instructions for setting up the Tensorflow serving environment, copying the model weights and configuration file into the container and commands in order to run the Tensorflow model server.

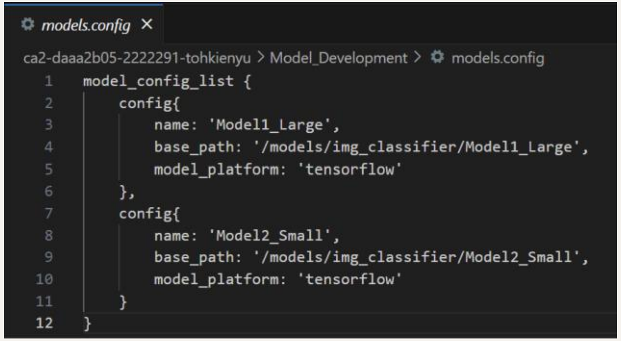
5. Model Development on Render
   Finally, I deployed the Docker container to Render.com. This ensures that the models were accessible via a web interface. Upon deployment, I conducted tests to ensure that the models were operational and responding as expected.

Dockerfile:


Dockerfile ×
ca2-daaa2b05-2222291-tohkienyu > Model_Development > Dockerfile

```
FROM tensorflow/serving

WORKDIR /Model_Development
COPY Model_Development/img_classifier/Model1_Large /models/img_classifier/Model1_Large
COPY Model_Development/img_classifier/Model2_Small /models/img_classifier/Model2_Small
COPY Model_Development/models.config /models/models.config

ENV MODEL_CONFIG_FILE=/models/models.config
EXPOSE 8501

CMD ["tensorflow_model_server","--rest_api_port=8501","--model_config_file=/models/models.config"]
```

models.config:


models.config ×
ca2-daaa2b05-2222291-tohkienyu > Model_Development > models.config

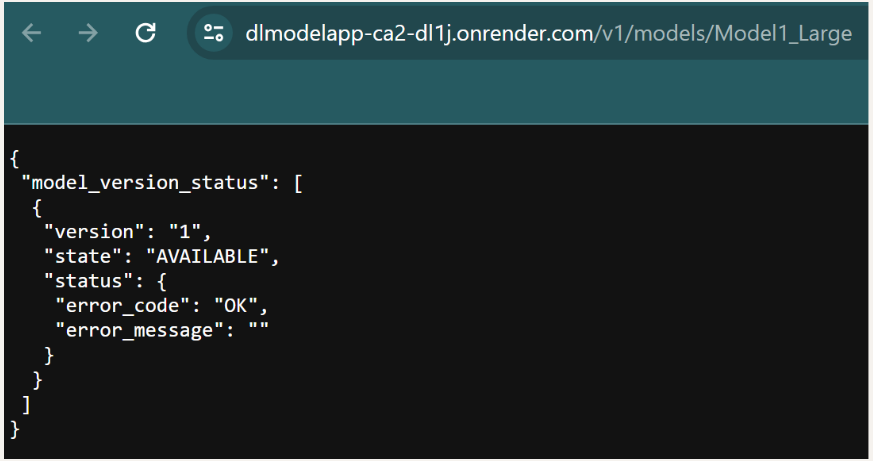
```
model_config_list {
    config{
        name: 'Model1_Large',
        base_path: '/models/img_classifier/Model1_Large',
        model_platform: 'tensorflow'
    },
    config{
        name: 'Model2_Small',
        base_path: '/models/img_classifier/Model2_Small',
        model_platform: 'tensorflow'
    }
}
```

Branch 1 (DLModel_branch)

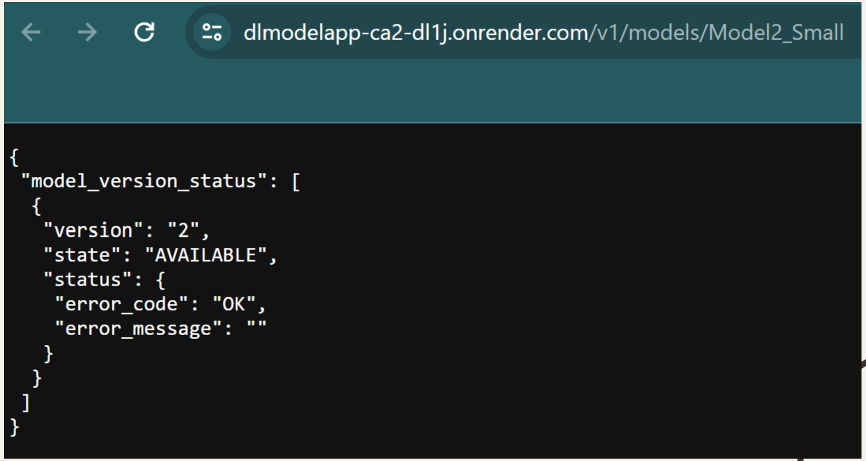
The model has been successfully deployed and now accessible at the following URL:

Model1_Large(128x128):
https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model1_Large

Model2_Small(31x31):
https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model2_Small



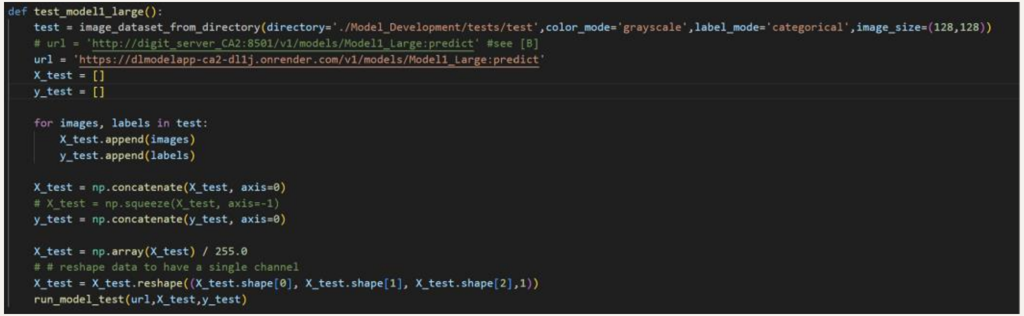
```
{
 "model_version_status": [
  {
   "version": "1",
   "state": "AVAILABLE",
   "status": {
    "error_code": "OK",
    "error_message": ""
   }
  }
 ]
}
```



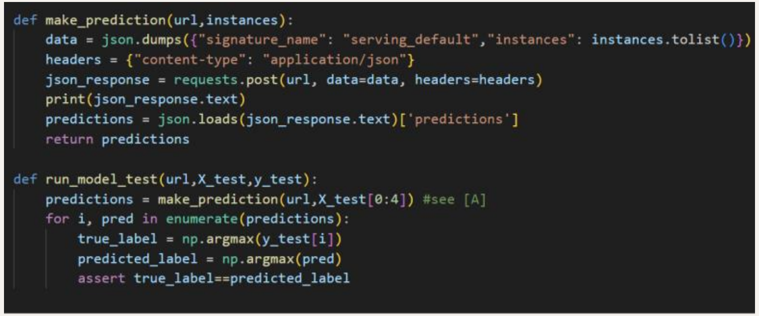
```
{
 "model_version_status": [
  {
   "version": "2",
   "state": "AVAILABLE",
   "status": {
    "error_code": "OK",
    "error_message": ""
   }
  }
 ]
}
```

Branch 1 (DLModel_branch)

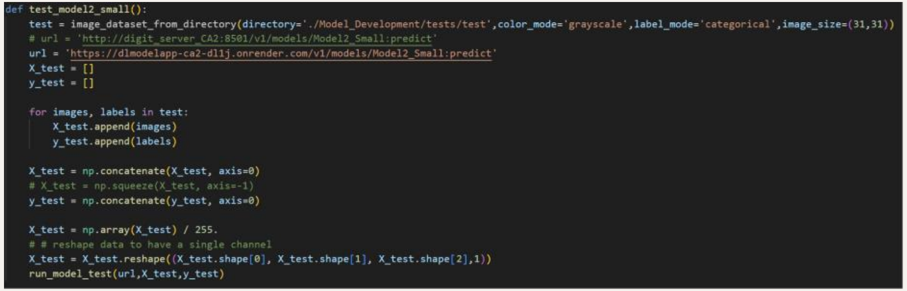
Testing was done to ensure both models were operational and responding as expected.



```python
def test_model1_large():
    test = image_dataset_from_directory(directory='./Model_Development/tests/test',color_mode='grayscale',label_mode='categorical',image_size=(128,128))
    # url = 'http://digit_server_CA2:8501/v1/models/Model1_Large:predict' #see [B]
    url = 'https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model1_Large:predict'
    X_test = []
    y_test = []

    for images, labels in test:
        X_test.append(images)
        y_test.append(labels)

    X_test = np.concatenate(X_test, axis=0)
    # X_test = np.squeeze(X_test, axis=-1)
    y_test = np.concatenate(y_test, axis=0)

    X_test = np.array(X_test) / 255.0
    # # reshape data to have a single channel
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], X_test.shape[2],1))
    run_model_test(url,X_test,y_test)
```

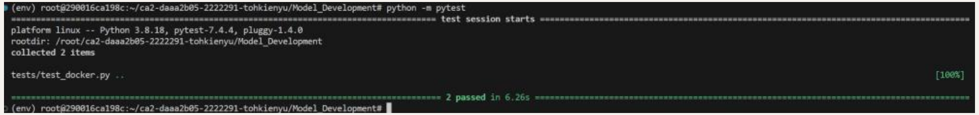


```python
def make_prediction(url,instances):
    data = json.dumps({"signature_name": "serving_default","instances": instances.tolist()})
    headers = {"content-type": "application/json"}
    json_response = requests.post(url, data=data, headers=headers)
    print(json_response.text)
    predictions = json.loads(json_response.text)['predictions']
    return predictions

def run_model_test(url,X_test,y_test):
    predictions = make_prediction(url,X_test[0:4]) #see [A]
    for i, pred in enumerate(predictions):
        true_label = np.argmax(y_test[i])
        predicted_label = np.argmax(pred)
        assert true_label==predicted_label
```



```python
def test_model2_small():
    test = image_dataset_from_directory(directory='./Model_Development/tests/test',color_mode='grayscale',label_mode='categorical',image_size=(31,31))
    # url = 'http://digit_server_CA2:8501/v1/models/Model2_Small:predict'
    url = 'https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model2_Small:predict'
    X_test = []
    y_test = []

    for images, labels in test:
        X_test.append(images)
        y_test.append(labels)

    X_test = np.concatenate(X_test, axis=0)
    # X_test = np.squeeze(X_test, axis=-1)
    y_test = np.concatenate(y_test, axis=0)

    X_test = np.array(X_test) / 255.
    # # reshape data to have a single channel
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], X_test.shape[2],1))
    run_model_test(url,X_test,y_test)
```

Test Results:



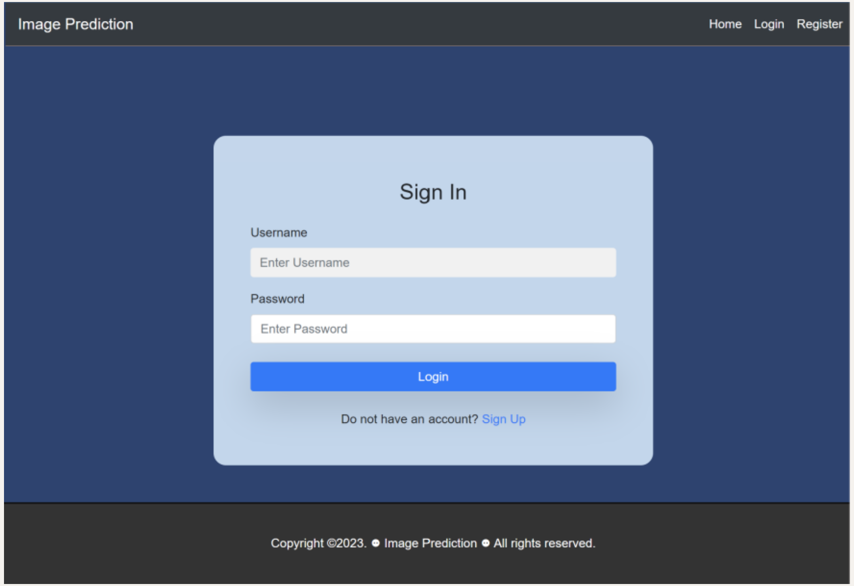
```
(env) root@290016ca198c:~/ca2-daaa2b05-2222291-tohkienyu/Model_Development# python -m pytest
============================================================ test session starts ============================================================
platform linux -- Python 3.8.18, pytest-7.4.4, pluggy-1.4.0
rootdir: /root/ca2-daaa2b05-2222291-tohkienyu/Model_Development
collected 2 items

tests/test_docker.py ..                                                                                                              [100%]

============================================================= 2 passed in 6.26s =============================================================
(env) root@290016ca198c:~/ca2-daaa2b05-2222291-tohkienyu/Model_Development# 
```

Branch 2 (DLWebApp_branch)

DLWebApp_branch focuses on integrating both deep learning models into the web application and enable users to upload an image before making a prediction.

This branch is also tasked on the designing of user interface which includes features such as home, login, registration, predict and prediction history page. Unit testing will also be done on this branch to ensure the web application's functionality and reliability.

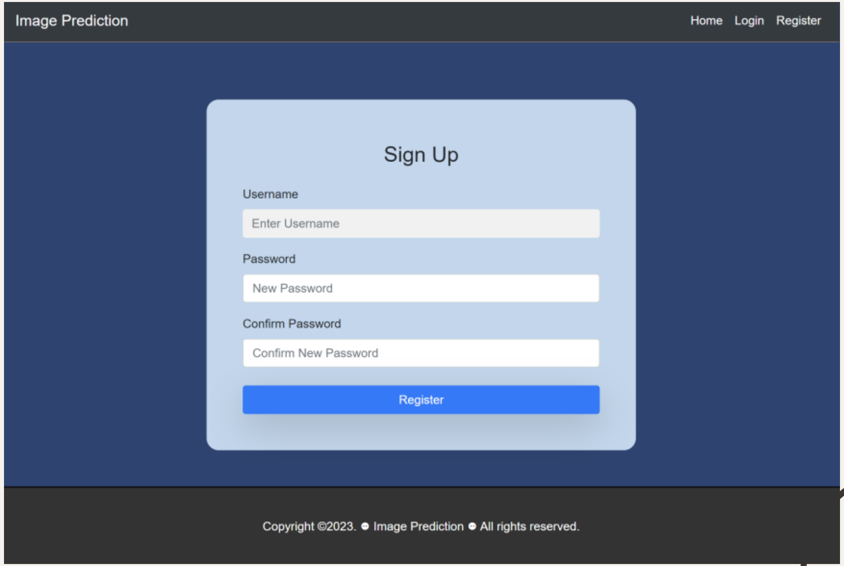
login.html



Do not have an account? Sign Up

Copyright ©2023. • Image Prediction • All rights reserved.

register.html

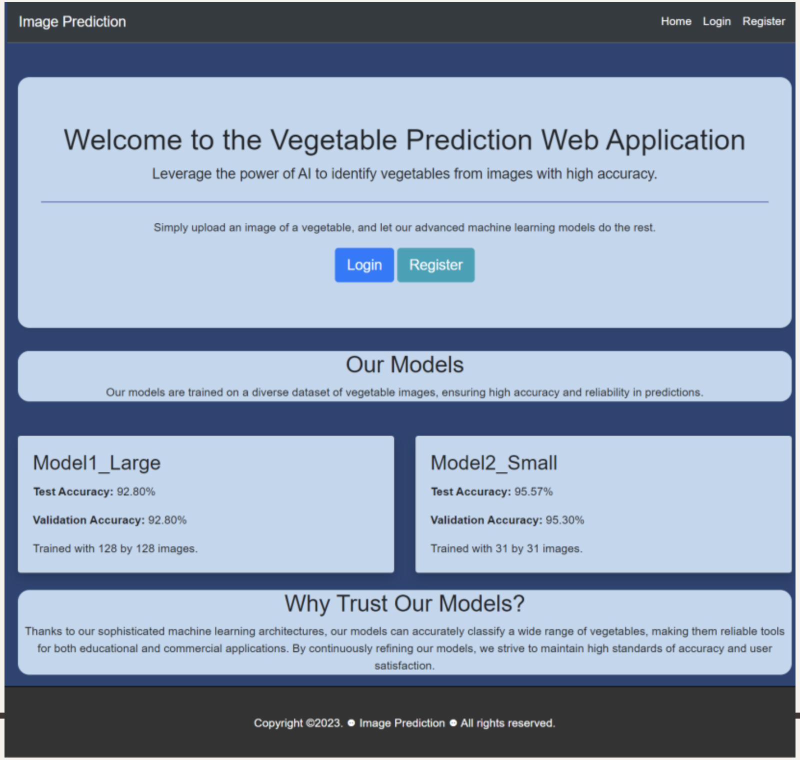


Copyright ©2023. • Image Prediction • All rights reserved.

Branch 2 (DLWebApp_branch)

predict.html allows user to switch between model 1 and model 2 of their choice before making a prediction

index.html



Welcome to the Vegetable Prediction Web Application

Leverage the power of AI to identify vegetables from images with high accuracy.

Simply upload an image of a vegetable, and let our advanced machine learning models do the rest.

Our Models

Our models are trained on a diverse dataset of vegetable images, ensuring high accuracy and reliability in predictions.

Model1_Large

Test Accuracy: 92.80%

Validation Accuracy: 92.80%

Trained with 128 by 128 images.

Model2_Small

Test Accuracy: 95.57%

Validation Accuracy: 95.30%

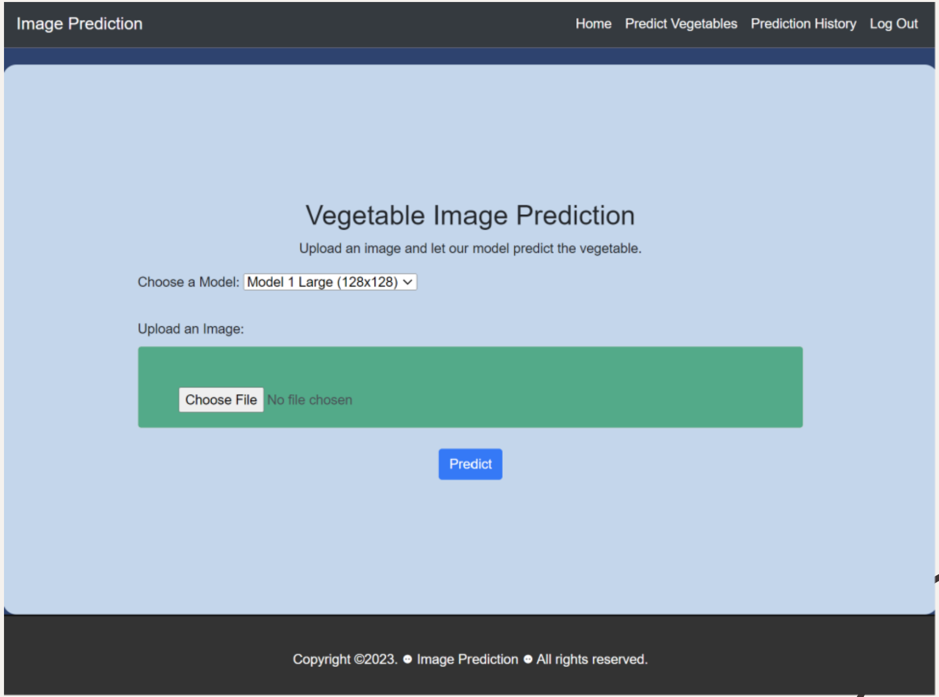
Trained with 31 by 31 images.

Why Trust Our Models?

Thanks to our sophisticated machine learning architectures, our models can accurately classify a wide range of vegetables, making them reliable tools for both educational and commercial applications. By continuously refining our models, we strive to maintain high standards of accuracy and user satisfaction.

Copyright ©2023. ● Image Prediction ● All rights reserved.

predict.html



Vegetable Image Prediction

Upload an image and let our model predict the vegetable.

Choose a Model: Model 1 Large (128x128)

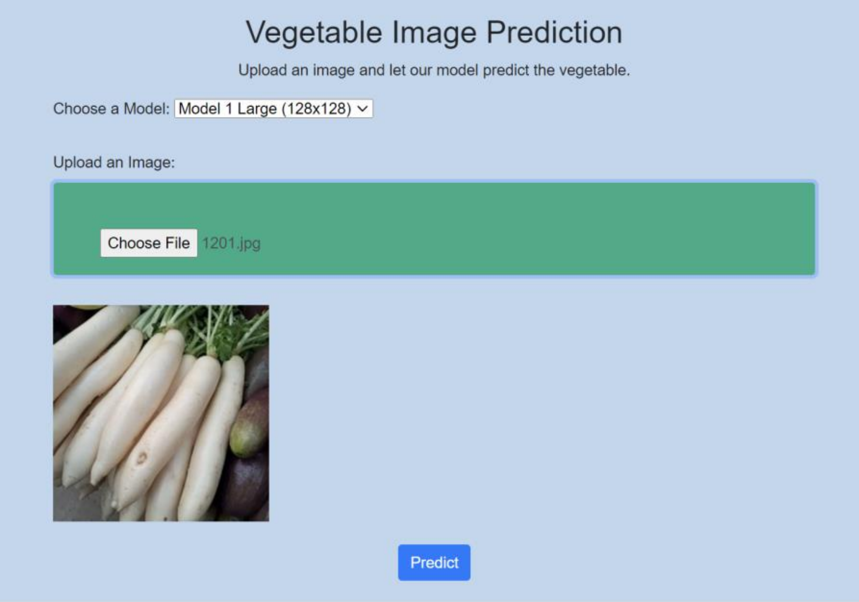
Upload an Image:

No file chosen

Copyright ©2023. ● Image Prediction ● All rights reserved.

Branch 2 (DLWebApp_branch)

For example, a user uploads a picture of a radish and press ‘Predict’ and it will return the prediction result and the confidence of the model’s prediction

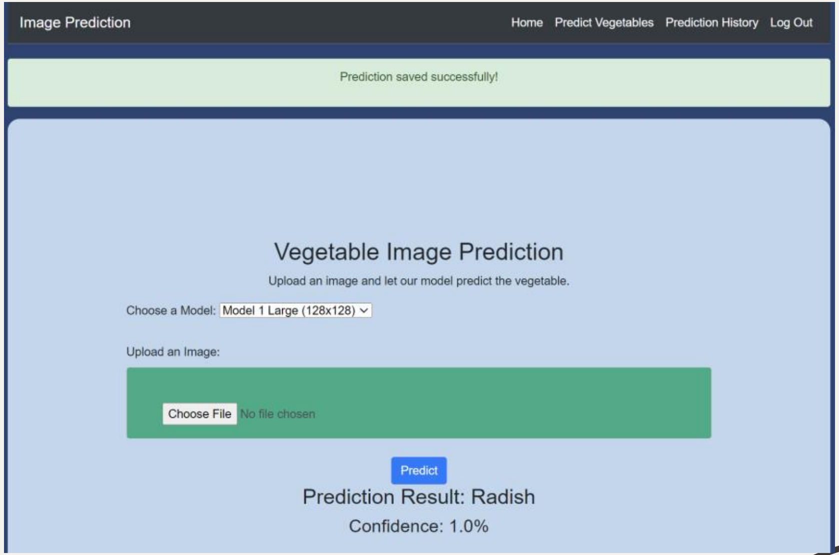


Vegetable Image Prediction

Upload an image and let our model predict the vegetable.

Choose a Model: Model 1 Large (128x128)

Upload an Image: 1201.jpg



Prediction saved successfully!

Vegetable Image Prediction

Upload an image and let our model predict the vegetable.

Choose a Model: Model 1 Large (128x128)

Upload an Image: No file chosen

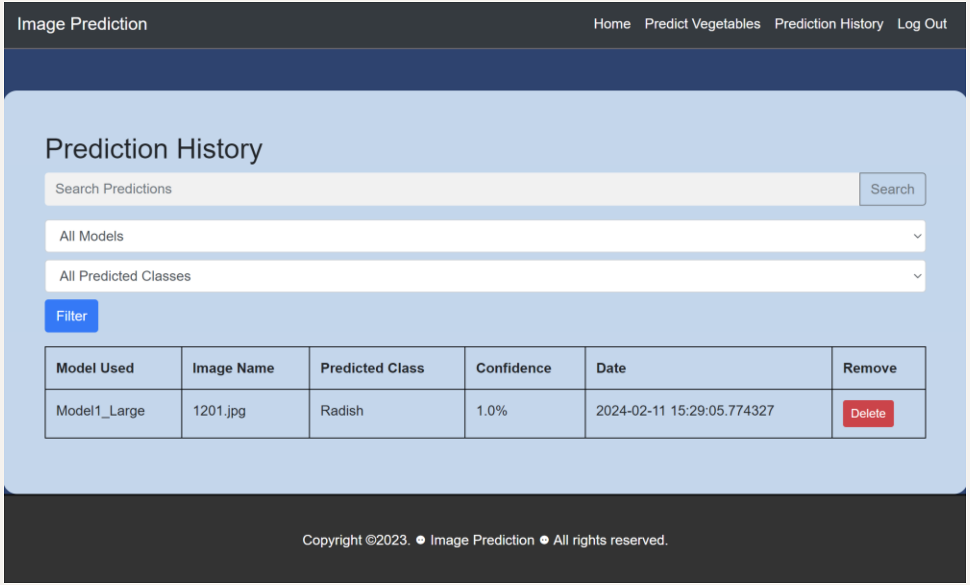
Prediction Result: Radish

Confidence: 1.0%

Branch 2 (DLWebApp_branch)

The prediction history page will record each predictions made by the user. Additionally, will then save the prediction. The prediction history page also offers capabilities for users to filter and search for results based on the model used and specific classes user wants to display through the use of the search and filter button. Users can also delete specific records.

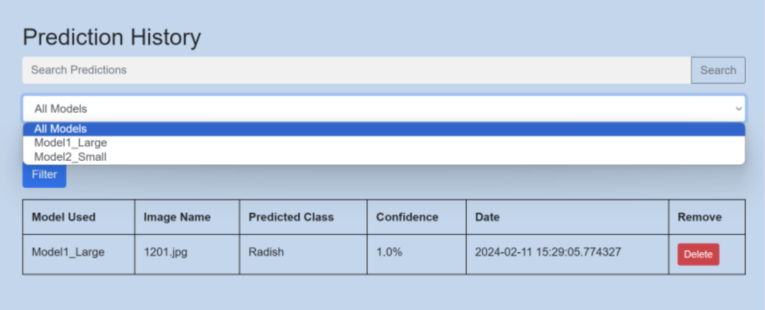
predictionHistory.html



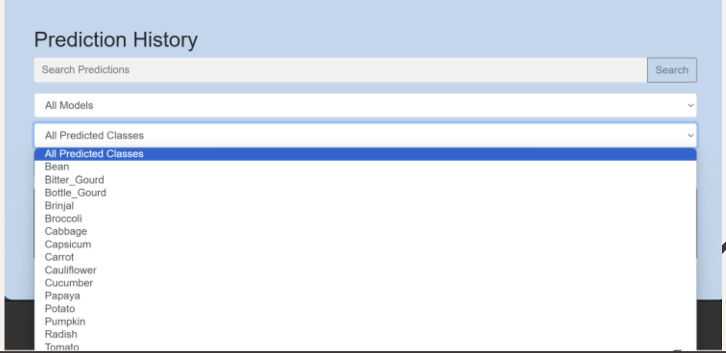
Prediction History

Model Used	Image Name	Predicted Class	Confidence	Date	Remove
Model1_Large	1201.jpg	Radish	1.0%	2024-02-11 15:29:05.774327	Delete

Copyright ©2023. • Image Prediction • All rights reserved.


Prediction History

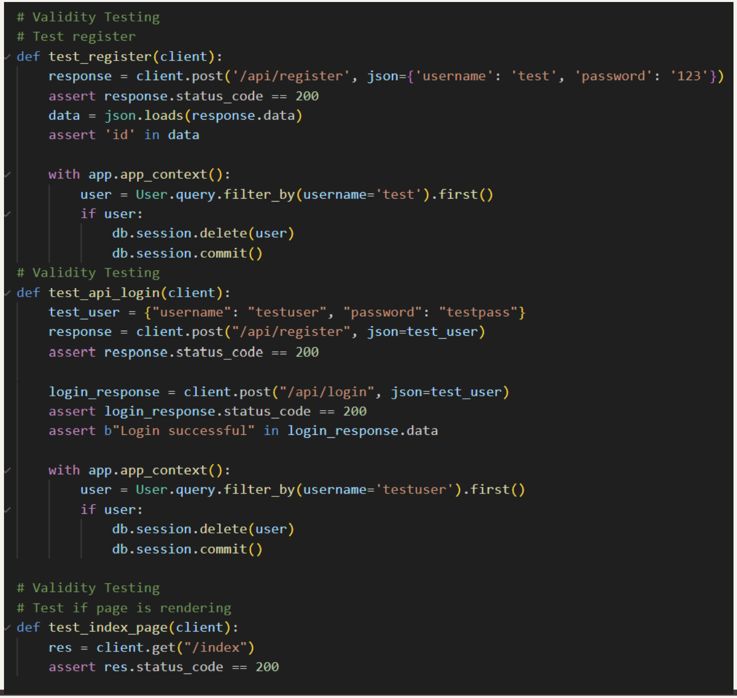
Model Used	Image Name	Predicted Class	Confidence	Date	Remove
Model1_Large	1201.jpg	Radish	1.0%	2024-02-11 15:29:05.774327	Delete


Prediction History

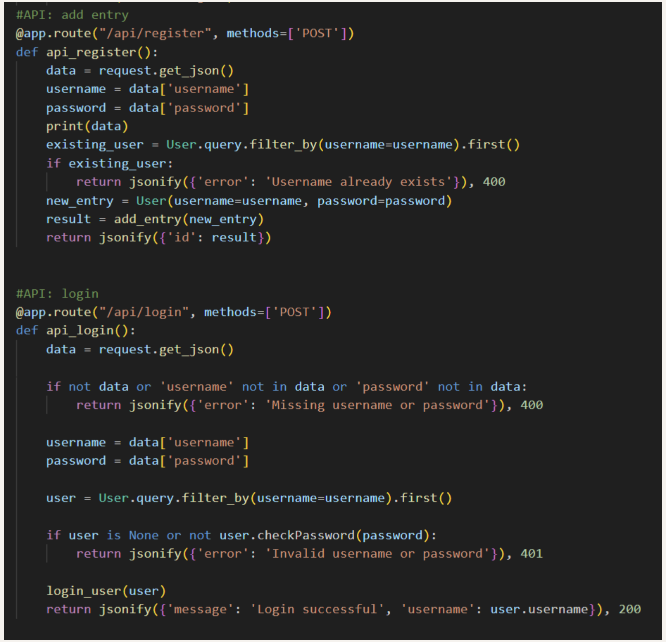
Branch 2 (DLWebApp_branch)

Unit Testing

Validity testing was done to make sure the registering and logging in of user is functioning correctly. This provides a seamless experience for users



```python
# Validity Testing
# Test register
def test_register(client):
    response = client.post('/api/register', json={'username': 'test', 'password': '123'})
    assert response.status_code == 200
    data = json.loads(response.data)
    assert 'id' in data

    with app.app_context():
        user = User.query.filter_by(username='test').first()
        if user:
            db.session.delete(user)
            db.session.commit()
# Validity Testing
def test_api_login(client):
    test_user = {"username": "testuser", "password": "testpass"}
    response = client.post("/api/register", json=test_user)
    assert response.status_code == 200

    login_response = client.post("/api/login", json=test_user)
    assert login_response.status_code == 200
    assert b"Login successful" in login_response.data

    with app.app_context():
        user = User.query.filter_by(username='testuser').first()
        if user:
            db.session.delete(user)
            db.session.commit()

# Validity Testing
# Test if page is rendering
def test_index_page(client):
    res = client.get("/index")
    assert res.status_code == 200
```

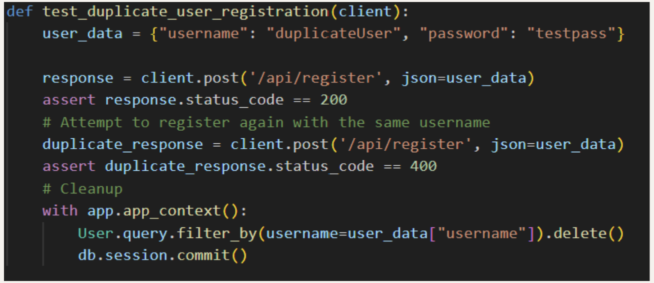


```python
#API: add entry
@app.route("/api/register", methods=['POST'])
def api_register():
    data = request.get_json()
    username = data['username']
    password = data['password']
    print(data)
    existing_user = User.query.filter_by(username=username).first()
    if existing_user:
        return jsonify({'error': 'Username already exists'}), 400
    new_entry = User(username=username, password=password)
    result = add_entry(new_entry)
    return jsonify({'id': result})


#API: login
@app.route("/api/login", methods=['POST'])
def api_login():
    data = request.get_json()

    if not data or 'username' not in data or 'password' not in data:
        return jsonify({'error': 'Missing username or password'}), 400

    username = data['username']
    password = data['password']

    user = User.query.filter_by(username=username).first()

    if user is None or not user.checkPassword(password):
        return jsonify({'error': 'Invalid username or password'}), 401

    login_user(user)
    return jsonify({'message': 'Login successful', 'username': user.username}), 200
```

Branch 2 (DLWebApp_branch)

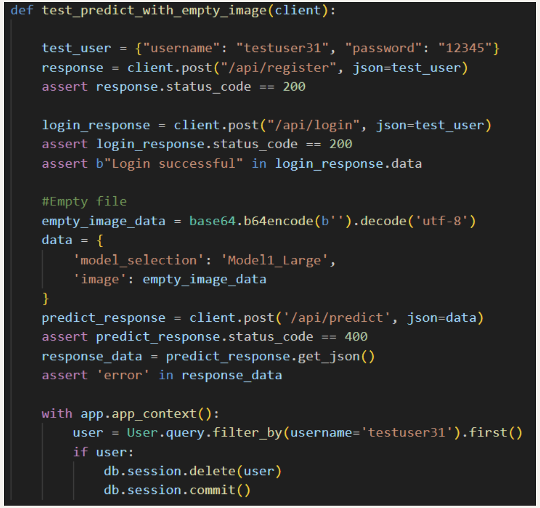
Unit Testing

Expected failure testing was done on Registration API to ensure the application handles duplicate username registration correctly.
- Passes when unique username is inserted into the database
- Expected failure when username is not unique

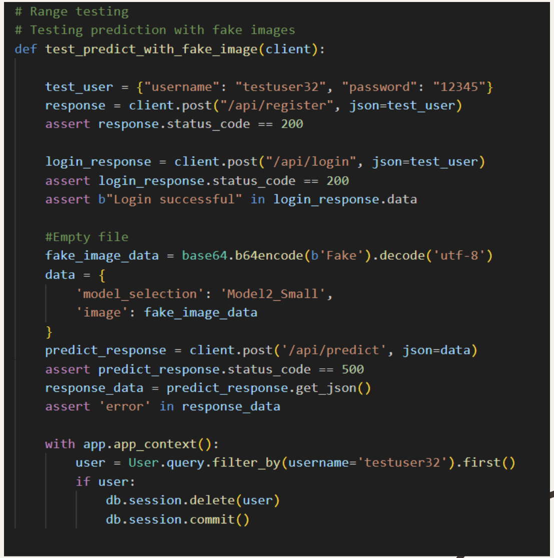


```
def test_duplicate_user_registration(client):
    user_data = {"username": "duplicateUser", "password": "testpass"}

    response = client.post('/api/register', json=user_data)
    assert response.status_code == 200
    # Attempt to register again with the same username
    duplicate_response = client.post('/api/register', json=user_data)
    assert duplicate_response.status_code == 400
    # Cleanup
    with app.app_context():
        User.query.filter_by(username=user_data["username"]).delete()
        db.session.commit()
```

Range testing was done to ensure that the web application can handle empty image or any invalid file given by user.



```
def test_predict_with_empty_image(client):

    test_user = {"username": "testuser31", "password": "12345"}
    response = client.post("/api/register", json=test_user)
    assert response.status_code == 200

    login_response = client.post("/api/login", json=test_user)
    assert login_response.status_code == 200
    assert b"Login successful" in login_response.data

    #Empty file
    empty_image_data = base64.b64encode(b'').decode('utf-8')
    data = {
        'model_selection': 'Model1_Large',
        'image': empty_image_data
    }
    predict_response = client.post('/api/predict', json=data)
    assert predict_response.status_code == 400
    response_data = predict_response.get_json()
    assert 'error' in response_data

    with app.app_context():
        user = User.query.filter_by(username='testuser31').first()
        if user:
            db.session.delete(user)
            db.session.commit()
```

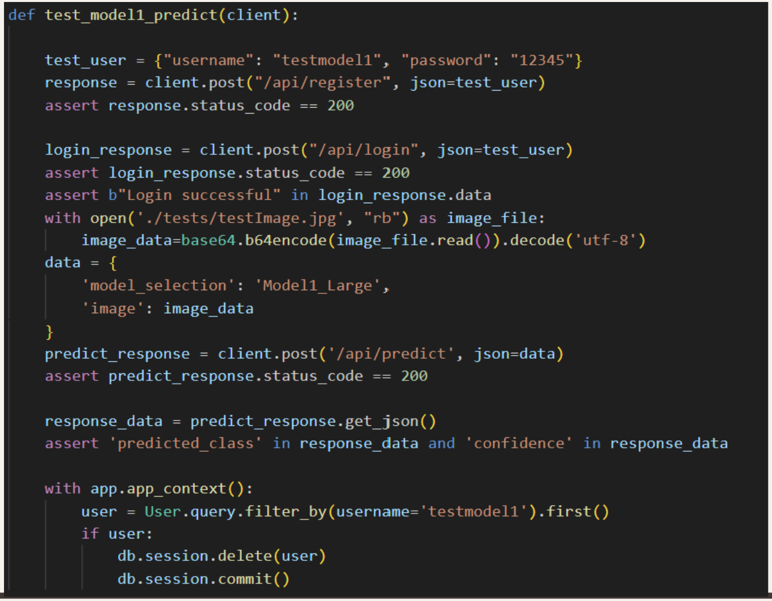


```
# Range testing
# Testing prediction with fake images
def test_predict_with_fake_image(client):

    test_user = {"username": "testuser32", "password": "12345"}
    response = client.post("/api/register", json=test_user)
    assert response.status_code == 200

    login_response = client.post("/api/login", json=test_user)
    assert login_response.status_code == 200
    assert b"Login successful" in login_response.data

    #Empty file
    fake_image_data = base64.b64encode(b'Fake').decode('utf-8')
    data = {
        'model_selection': 'Model2_Small',
        'image': fake_image_data
    }
    predict_response = client.post('/api/predict', json=data)
    assert predict_response.status_code == 500
    response_data = predict_response.get_json()
    assert 'error' in response_data

    with app.app_context():
        user = User.query.filter_by(username='testuser32').first()
        if user:
            db.session.delete(user)
            db.session.commit()
```

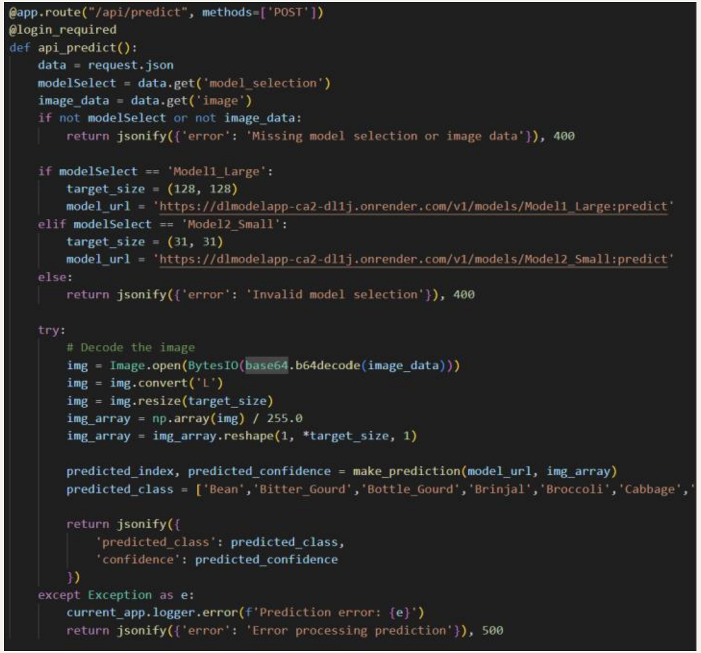
Branch 2 (DLWebApp_branch)

Unit Testing

Validity and **Consistency testing** was done to ensure the model operates correctly. This tests ensure that both model are able to consistently correctly produce predictions, thus verifying the application’s functionality in handling prediction requests accurately.

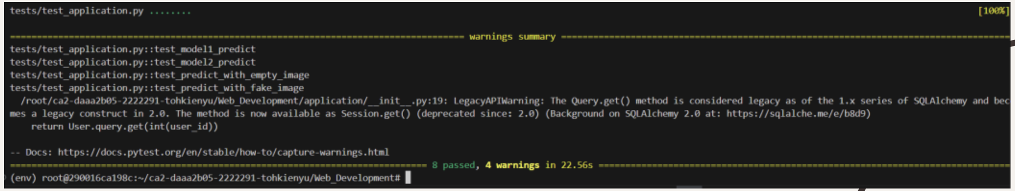


```
def test_model1_predict(client):

    test_user = {"username": "testmodel1", "password": "12345"}
    response = client.post("/api/register", json=test_user)
    assert response.status_code == 200

    login_response = client.post("/api/login", json=test_user)
    assert login_response.status_code == 200
    assert b"Login successful" in login_response.data
    with open('./tests/testImage.jpg', "rb") as image_file:
        image_data=base64.b64encode(image_file.read()).decode('utf-8')
    data = {
        'model_selection': 'Model1_Large',
        'image': image_data
    }
    predict_response = client.post('/api/predict', json=data)
    assert predict_response.status_code == 200

    response_data = predict_response.get_json()
    assert 'predicted_class' in response_data and 'confidence' in response_data

    with app.app_context():
        user = User.query.filter_by(username='testmodel1').first()
        if user:
            db.session.delete(user)
            db.session.commit()
```



```
@app.route("/api/predict", methods=['POST'])
@login_required
def api_predict():
    data = request.json
    modelSelect = data.get('model_selection')
    image_data = data.get('image')
    if not modelSelect or not image_data:
        return jsonify({'error': 'Missing model selection or image data'}), 400

    if modelSelect == 'Model1_Large':
        target_size = (128, 128)
        model_url = 'https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model1_Large:predict'
    elif modelSelect == 'Model2_Small':
        target_size = (31, 31)
        model_url = 'https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model2_Small:predict'
    else:
        return jsonify({'error': 'Invalid model selection'}), 400

    try:
        # Decode the image
        img = Image.open(BytesIO(base64.b64decode(image_data)))
        img = img.convert('L')
        img = img.resize(target_size)
        img_array = np.array(img) / 255.0
        img_array = img_array.reshape(1, *target_size, 1)

        predicted_index, predicted_confidence = make_prediction(model_url, img_array)
        predicted_class = ['Bean','Bitter_Gourd','Bottle_Gourd','Brinjal','Broccoli','Cabbage','

        return jsonify({
            'predicted_class': predicted_class,
            'confidence': predicted_confidence
        })
    except Exception as e:
        current_app.logger.error(f'Prediction error: {e}')
        return jsonify({'error': 'Error processing prediction'}), 500
```

Summary of tests

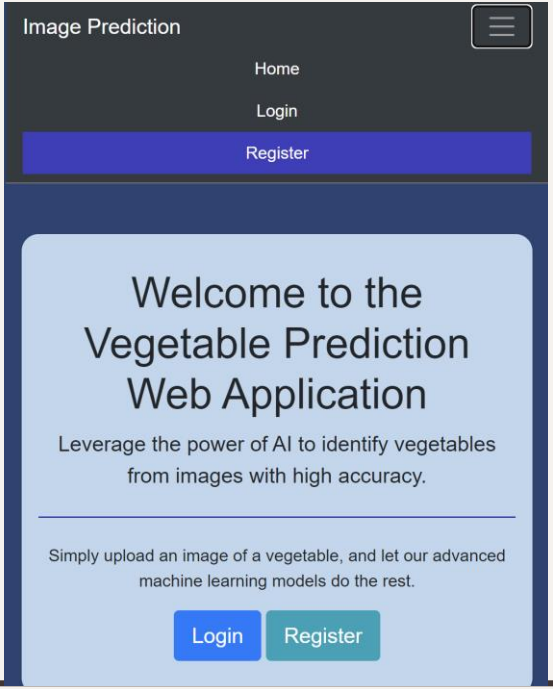


```
tests/test_application.py ........                                                                                    [100%]

=============================================== warnings summary ===============================================
tests/test_application.py::test_model1_predict
tests/test_application.py::test_model2_predict
tests/test_application.py::test_predict_with_empty_image
tests/test_application.py::test_predict_with_fake_image
  /root/ca2-daaa2b05-2222291-tohkienyu/Web_Development/application/__init__.py:19: LegacyAPIWarning: The Query.get() method is considered legacy as of the 1.x series of SQLAlchemy and bec
mes a legacy construct in 2.0. The method is now available as Session.get() (deprecated since: 2.0) (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
    return User.query.get(int(user_id))

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========================================== 8 passed, 4 warnings in 22.56s ==========================================
(env) root@290016ca198c:~/ca2-daaa2b05-2222291-tohkienyu/Web_Development#
```

Branch 3 (improveUI_branch)

- Responsive Web Application (Catered for smaller and larger devices)

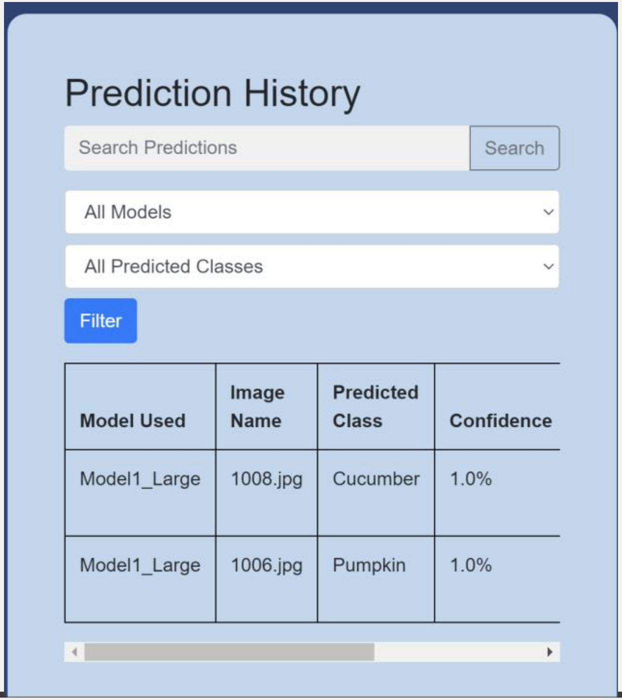
For example: Dropdown menu was added for smaller devices and hover animations. History table was also made responsive with scrollbar added.



Welcome to the Vegetable Prediction Web Application

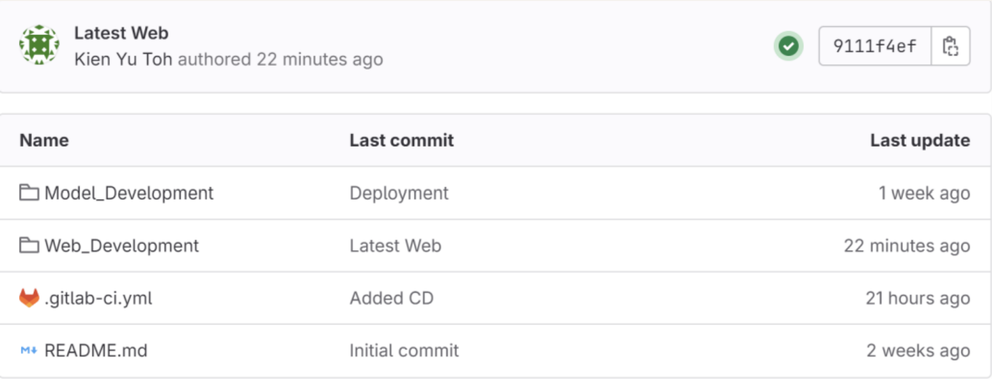
Leverage the power of AI to identify vegetables from images with high accuracy.

Simply upload an image of a vegetable, and let our advanced machine learning models do the rest.


Prediction History

Model Used	Image Name	Predicted Class	Confidence
Model1_Large	1008.jpg	Cucumber	1.0%
Model1_Large	1006.jpg	Pumpkin	1.0%

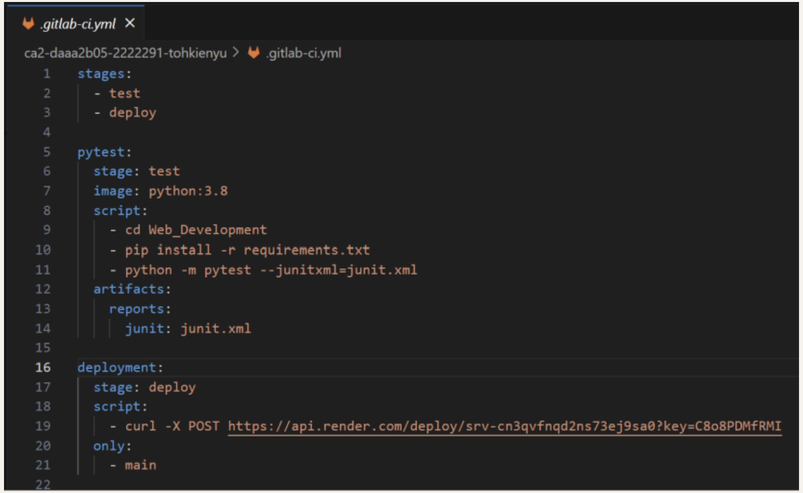
Branch 4 (CICD_branch)



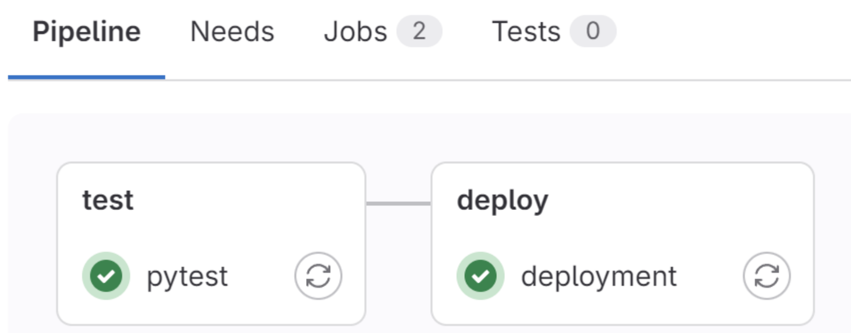
Latest Web
Kien Yu Toh authored 22 minutes ago

9111f4ef

Name	Last commit	Last update
Model_Development	Deployment	1 week ago
Web_Development	Latest Web	22 minutes ago
.gitlab-ci.yml	Added CD	21 hours ago
README.md	Initial commit	2 weeks ago



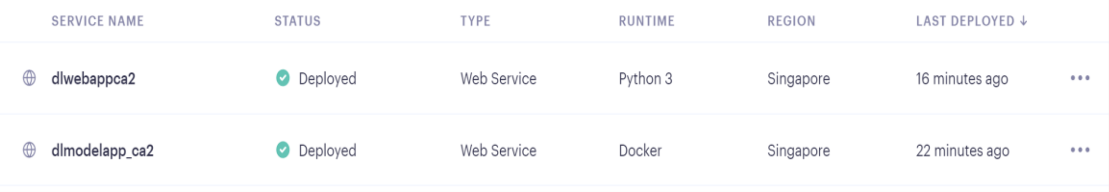
```yaml
stages:
  - test
  - deploy

pytest:
  stage: test
  image: python:3.8
  script:
    - cd Web_Development
    - pip install -r requirements.txt
    - python -m pytest --junitxml=junit.xml
  artifacts:
    reports:
      junit: junit.xml

deployment:
  stage: deploy
  script:
    - curl -X POST https://api.render.com/deploy/srv-cn3qvfnqd2ns73ej9sa0?key=C8o8PDMfRMI
  only:
    - main

```



Branch 4 integrates the CI/CD pipeline configuration to automate the testing and deployment process. This setup ensures that every change is automatically tested, and builds are successful before it gets deployed which ensures a efficient workflow.

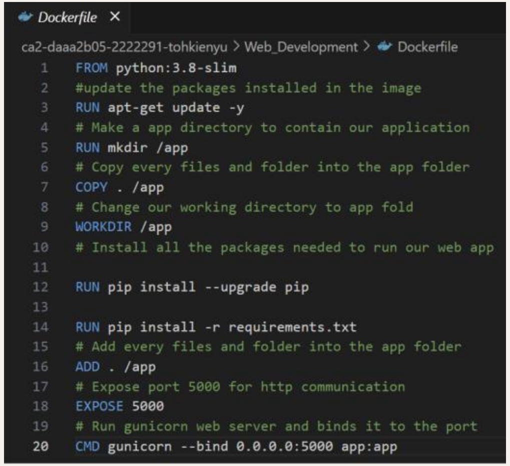
Internet Deployment

Render was used to deploy the web application



SERVICE NAME	STATUS	TYPE	RUNTIME	REGION	LAST DEPLOYED ↓	
dlwebappca2	Deployed	Web Service	Python 3	Singapore	16 minutes ago	•••
dlmodelapp_ca2	Deployed	Web Service	Docker	Singapore	22 minutes ago	•••

Dockerfile for web:

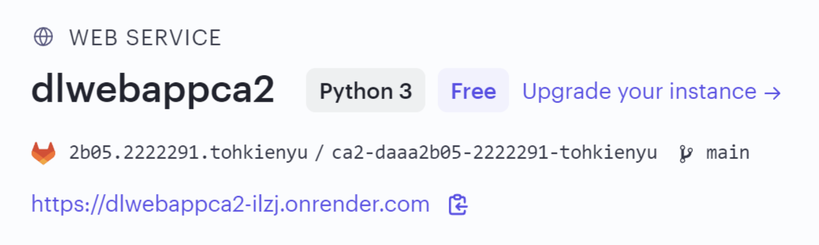


Dockerfile ×
ca2-daaa2b05-2222291-tohkienyu > Web_Development > Dockerfile

```dockerfile
FROM python:3.8-slim
#update the packages installed in the image
RUN apt-get update -y
# Make a app directory to contain our application
RUN mkdir /app
# Copy every files and folder into the app folder
COPY . /app
# Change our working directory to app fold
WORKDIR /app
# Install all the packages needed to run our web app

RUN pip install --upgrade pip

RUN pip install -r requirements.txt
# Add every files and folder into the app folder
ADD . /app
# Expose port 5000 for http communication
EXPOSE 5000
# Run gunicorn web server and binds it to the port
CMD gunicorn --bind 0.0.0.0:5000 app:app
```

Final Web Application:
https://dlwebappca2-ilzj.onrender.com/

Models:
https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model1_Large
https://dlmodelapp-ca2-dl1j.onrender.com/v1/models/Model2_Small

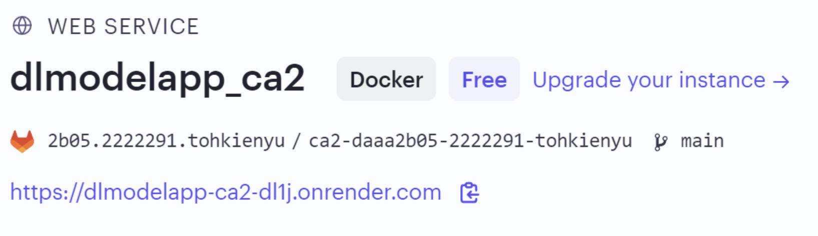


WEB SERVICE

dlwebappca2 Python 3 Free Upgrade your instance →

2b05.2222291.tohkienyu / ca2-daaa2b05-2222291-tohkienyu main

https://dlwebappca2-ilzj.onrender.com



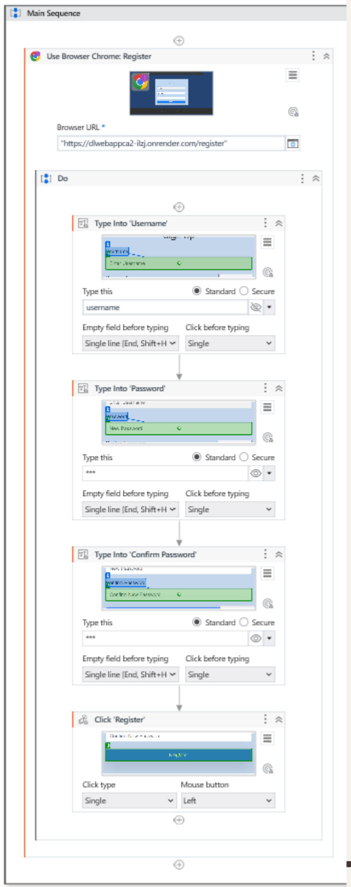
WEB SERVICE

dlmodelapp_ca2 Docker Free Upgrade your instance →

2b05.2222291.tohkienyu / ca2-daaa2b05-2222291-tohkienyu main

https://dlmodelapp-ca2-dl1j.onrender.com

RPA



Main Sequence

Use Browser Chrome: Register

Browser URL *
"https://dlwebappca2-ilzj.onrender.com/register"

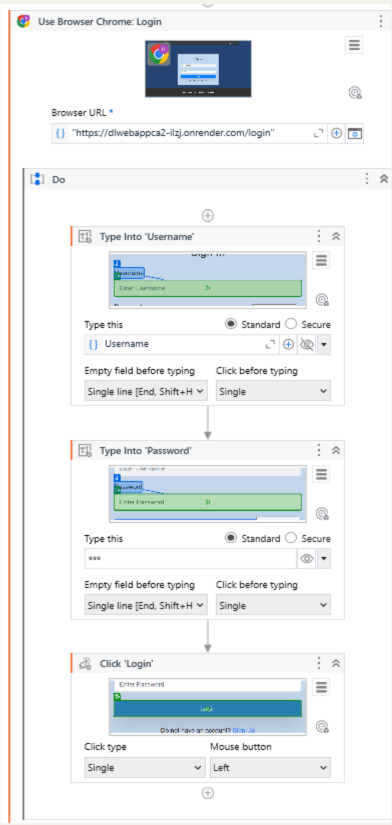
Do

Type Into 'Username'
Type this ● Standard ○ Secure
username
Empty field before typing: Single line (End, Shift+H
Click before typing: Single

Type Into 'Password'
Type this ● Standard ○ Secure
***
Empty field before typing: Single line (End, Shift+H
Click before typing: Single

Type Into 'Confirm Password'
Type this ● Standard ○ Secure
***
Empty field before typing: Single line (End, Shift+H
Click before typing: Single

Click 'Register'
Click type: Single
Mouse button: Left



Use Browser Chrome: Login

Browser URL *
{} "https://dlwebappca2-ilzj.onrender.com/login"

Do

Type Into 'Username'
Type this ● Standard ○ Secure
{} Username
Empty field before typing: Single line (End, Shift+H
Click before typing: Single

Type Into 'Password'
Type this ● Standard ○ Secure
***
Empty field before typing: Single line (End, Shift+H
Click before typing: Single

Click 'Login'
Click type: Single
Mouse button: Left

UIPath was used to automate:

1. Logging in of credentials
2. Registering of new users

This allows us to execute login and register sequence automatically and reduces these repetitive and time consuming tasks

Thank You

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**