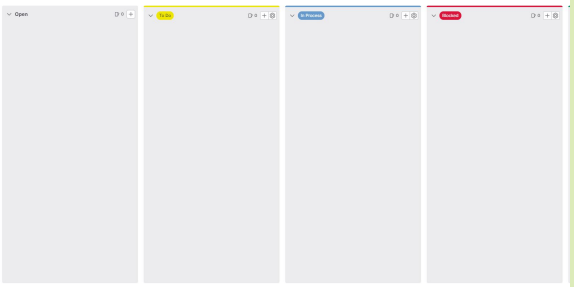
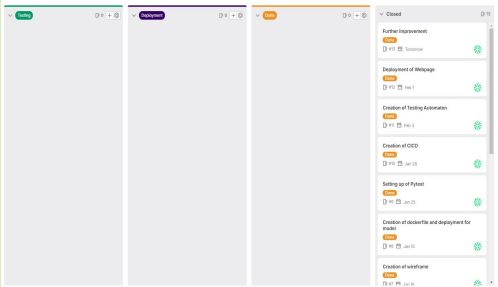
VG-15 APP

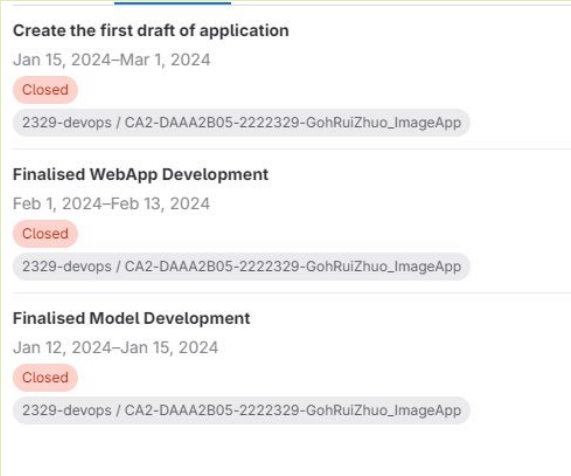
Done By: Goh Rui Zhuo (2222329)

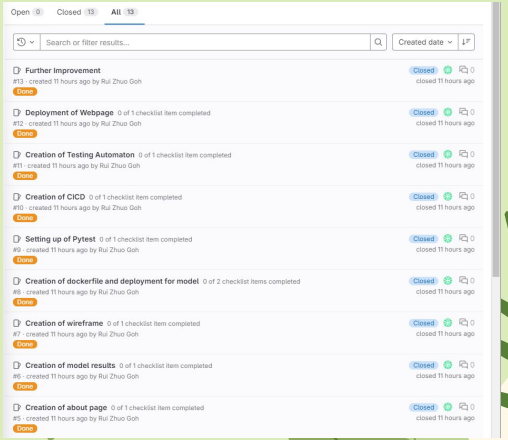
Scrum Board

1. Added six section which are
   a. To Do
   b. In Process
   c. Blocked
   d. Testing
   e. Deployment
   f. Done
2. Milestones created (Sprint)
3. 13 overall issue boards









Branches Created

1. 6 separate branches were created

main default d8b9316c · Merge branch 'FurtherImprove_Development' into 'main' · 15 minutes ago		
FurtherImprove_Development d8e237d1 · final changes · 16 minutes ago	73	0
Wireframe_Development 934594e5 · Update History Page.drawio · 12 hours ago	31	0
CICD_Development 8133445e · changes · 1 day ago	93	0
Testing_App 6a2c3851 · add webapp · 1 day ago	106	0
Web_Development 6a2c3851 · add webapp · 1 day ago	106	0
Model_Development feb250ab · added model development · 1 day ago	107	0

Image Model Development

1. **All models and webapp are done in docker container

	DLModels_Server 1818fccb6089	python:3.8	Running	1.25%	18 hours ago	

Model Creation

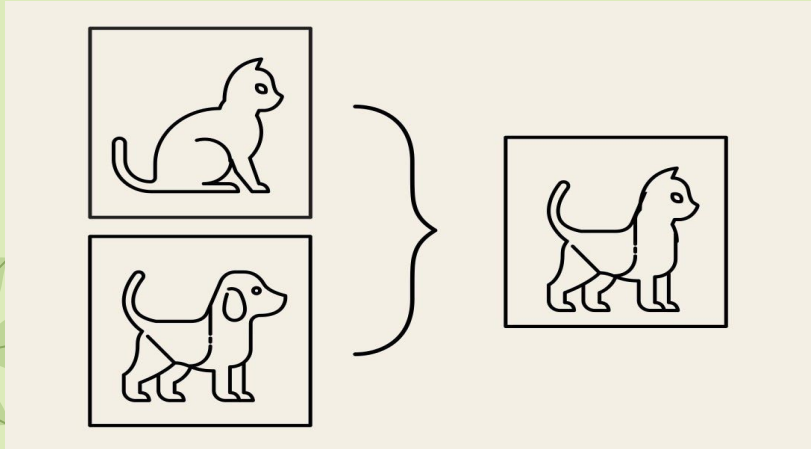
1. Baseline Dense Model (31 x 31 model)
2. CNN Baseline 1 Model (31 x 31 model)
3. CNN Baseline 2 Model (128 x 128 model)
    a. Initial Convolutional Block
    b. Three more convolutional block
    c. 1 dense layer
4. Custom VGG 16 model (128 x 128 model)
    a. Increase of layers
    b. Starts at a lower number of filters
    c. Change in conv block (reduce to 3 from 5)
    d. Added regularisation

And many others…

Image Model Improvement

Special Techniques for data augmentation

1. CutMix
   a. Cut and paste random patches between the training images/
2. CutOut
   a. Cut out certain parts of the image and input black parts onto it
3. Image Data Generator
   a. Random rotating here



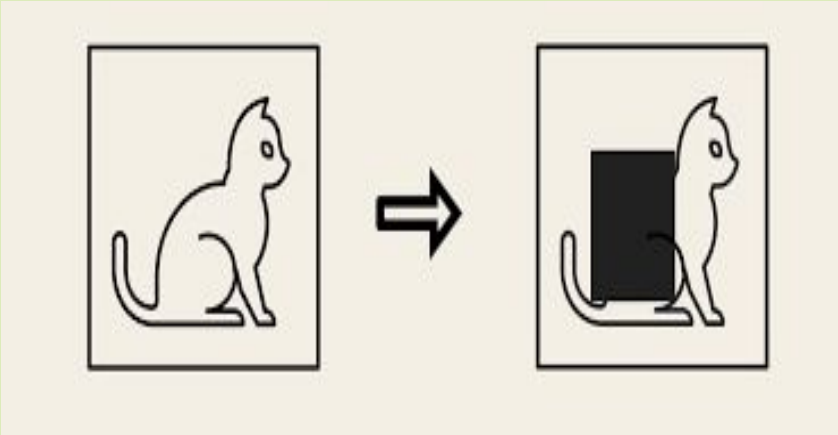


Image Model Improvement

Learning Rate Scheduler

1. Idea is to adjust the learning rate as the iterations increases
2. Start out with relatively high learning rates for several iterations in the beginning to quickly approach a local minimum
   a. Then gradually decrease the learning rate as we get closer to the minimum, ending with several small learning rate iterations
3. Goal of neural network is to find the minimum of the loss function hence this is important to not let the model skip the minimum point

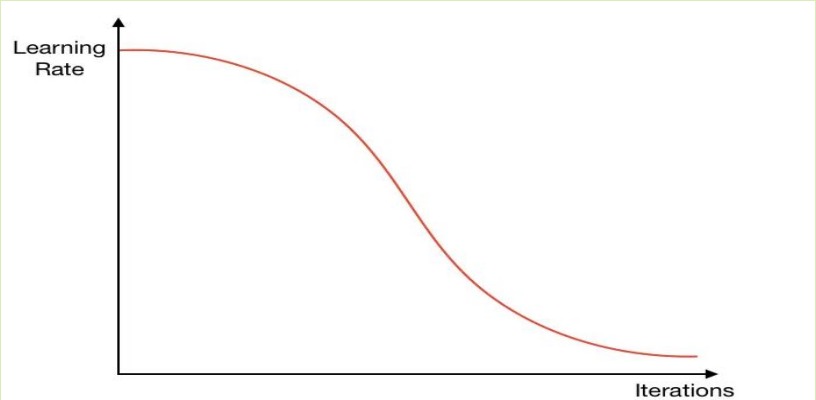


Image Model Improvement

1. 128 x 128 and 31 x 31 Images (Auto Encoder)

1. With autoencoder, we try to remove the outliers from the dataset
2. Below shows the separation of the data and only non outliers was use in training of model
3. However, for both image size, the validation score did not improve likely due to the images being very clear and concise

```
X_train_small_expanded = tf.expand_dims(X_train_small, -1)
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, InputLayer,
# Encoder
encoder = Sequential([
    InputLayer(input_shape=X_train_small_expanded.shape[1:]),
    Conv2D(16, (3, 3), activation='relu', padding='same', strides=2),
    Conv2D(8, (3, 3), activation='relu', padding='same', strides=2)
])

# Decoder
decoder = Sequential([
    Conv2DTranspose(8, kernel_size=3, strides=2, activation='relu', paddi
    Conv2DTranspose(16, kernel_size=3, strides=2, activation='relu', padd
    Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same'),
    Cropping2D(cropping=((1, 0), (1, 0)))
])

autoencoder = Sequential([encoder, decoder])

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

history = autoencoder.fit(
    X_train_small_expanded, X_train_small_expanded,
    epochs=100,
    batch_size=128,
    validation_split=0.1
)
```

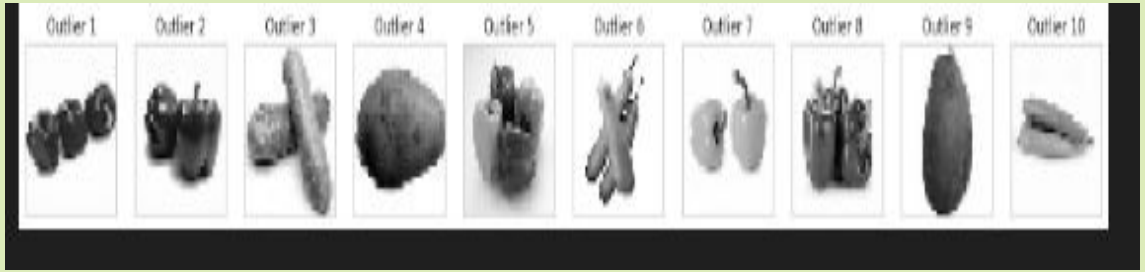


Image Model (Final)

31 x 31 Images

1. **Best Model:** CNN1 with cutmix
2. Achieved 97.2 validation and test accuracy
3. Model is best at prediction Bottle Gourd and worst at Potato in terms of f1 score
4. Cohen Kappa score is also at 97%

128 x 128 Images

1. **Best Model:** CNN2 with cutmix
2. Achieved 99.2 validation and 98.8 test accuracy
3. Model is best at prediction Bottle Gourd and worst at Radish in terms of f1 score
4. Cohen Kappa score is also at 98.75%

Model (Final with Deployment)

31 x 31 Images

1. **Best Model:** CNN1 with cutmix
2. Achieved 97.2 validation and test accuracy
3. Model is best at prediction Bottle Gourd and worst at Potato in terms of f1 score
4. Cohen Kappa score is also at 97%

dl-model-o6iu.onrender.com/v1/models/small


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


128 x 128 Images

1. **Best Model:** CNN2 with cutmix
2. Achieved 99.2 validation and 98.8 test accuracy
3. Model is best at prediction Bottle Gourd and worst at Radish in terms of f1 score
4. Cohen Kappa score is also at 98.75%

WEB SERVICE

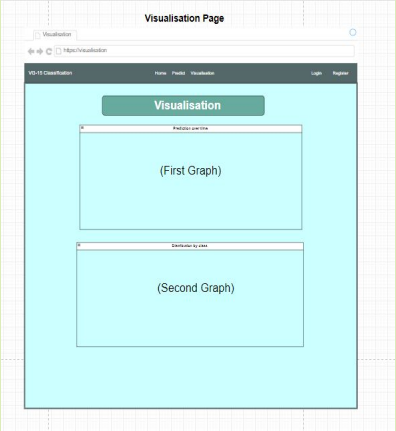
dl_model Docker Free Upgrade your instance →

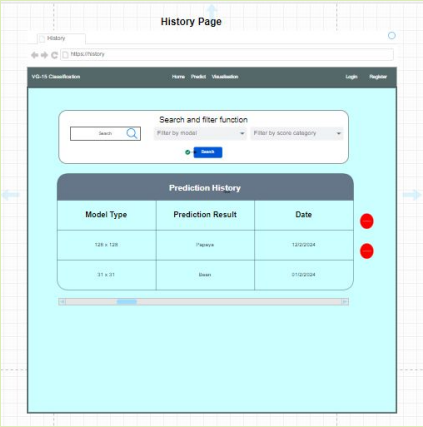
dl-model-o6iu.onrender.com/v1/models/big

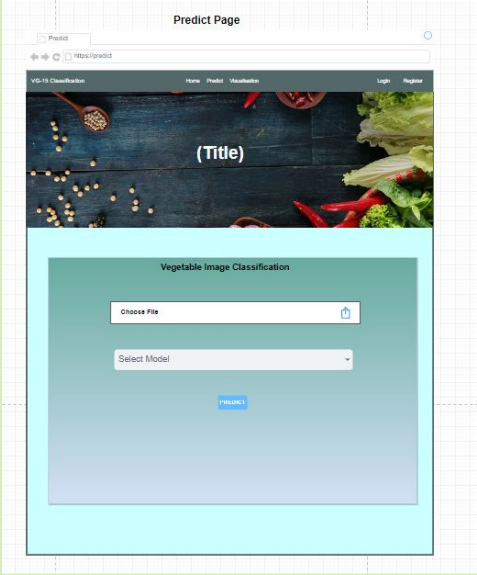

```
{
 "model_version_status": [
  {
   "version": "1",
   "state": "AVAILABLE",
   "status": {
    "error_code": "OK",
    "error_message": ""
   }
  }
 ]
}
```

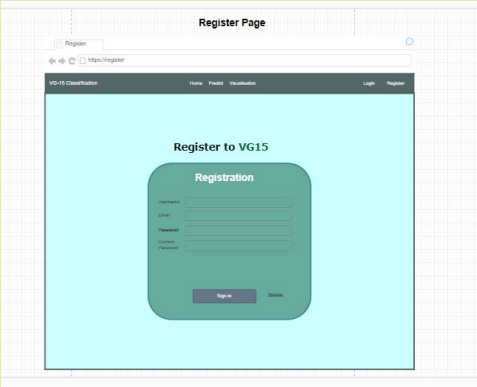
Wireframes

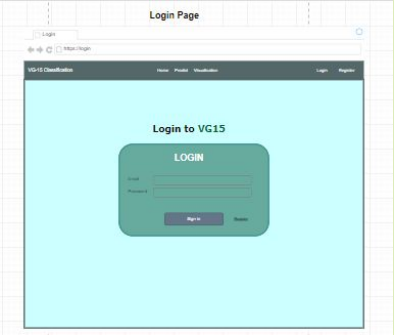
1. Clean and concise design were used here







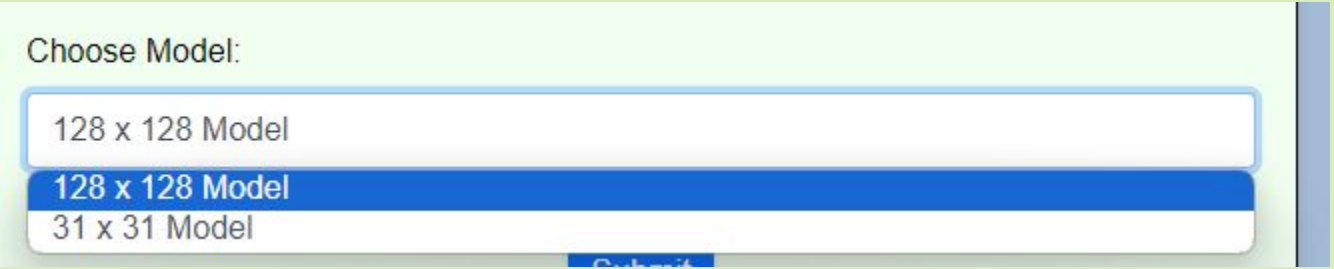




Deploying two Models

1. Deploy two models for user
2. Models can be switch with the use of dropdown menu

```
    return redirect(url_for('predict'))

if model_choice == 'big':
    image = tf.image.resize(image, [128, 128])
    model_url = 'https://dl-model-o6iu.onrender.com/v1/models/big:predict'
else:
    image = tf.image.resize(image, [31, 31])
    model_url = 'https://dl-model-o6iu.onrender.com/v1/models/small:predict'
```

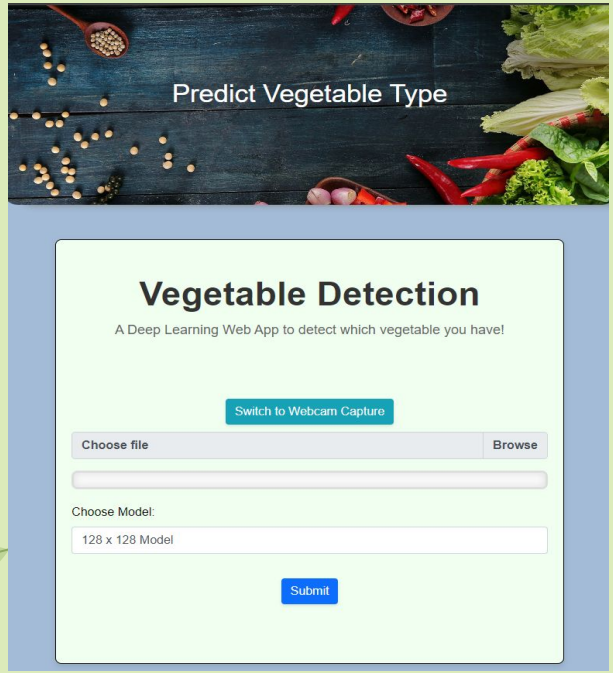


Web Development (Home)

1. Home page design with the use of vegetable images here
2. Added an introduction
3. Provided idea of our model

Web Development (Predict)

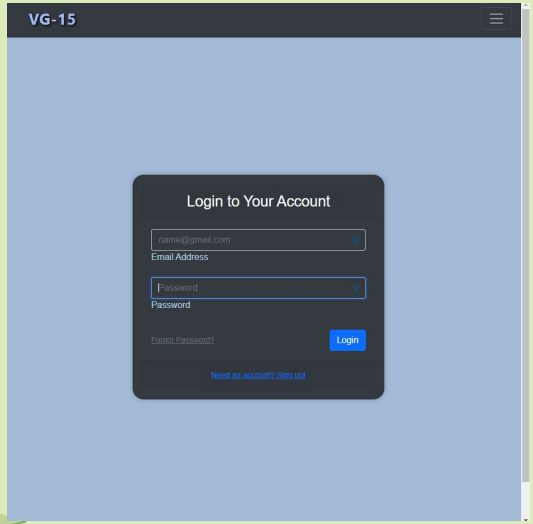
1. Prediction page was designed with the idea of providing user to seamless switch between models
2. Provide user input which is the upload of models here
3. Deployed with the use of flask framework

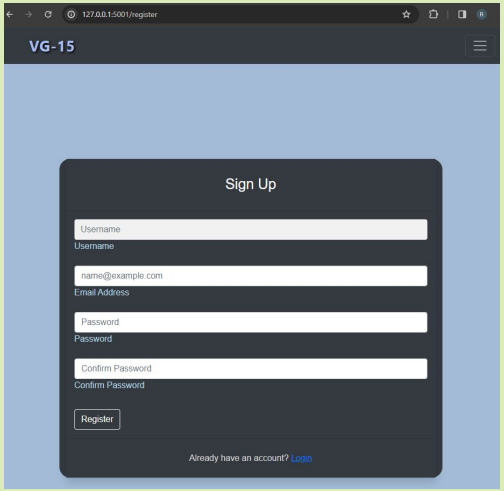



```python
class Prediction(db.Model):
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    userid = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    filename = db.Column(db.String, nullable=False)
    model = db.Column(db.String, nullable=False)
    predicted_on = db.Column(db.DateTime, nullable=False)
    prediction = db.Column(db.String, nullable=False)
    confidence_score = db.Column(db.Float, nullable=True)
    confidence_cat = db.Column(db.String, nullable=True)

    @validates('filename', 'model', 'prediction', 'confidence_cat')
    def validate_strings(self, key, value):
        assert value != '', f'{key} cannot be empty'
        return value

    @validates('predicted_on')
    def validate_predicted_on(self, key, value):
        assert isinstance(value, datetime), f'{key} must be a datetime object'
        return value

    @validates('confidence_score')
    def validate_confidence(self, key, value):
```

Web Development (Login and Sign up)

1. Login Page provides a clean and responsive look
2. It provides additional option for password resetting and all
3. Provides redirecting to register page if user has not log in before
4. Deployed with flask framework
5.

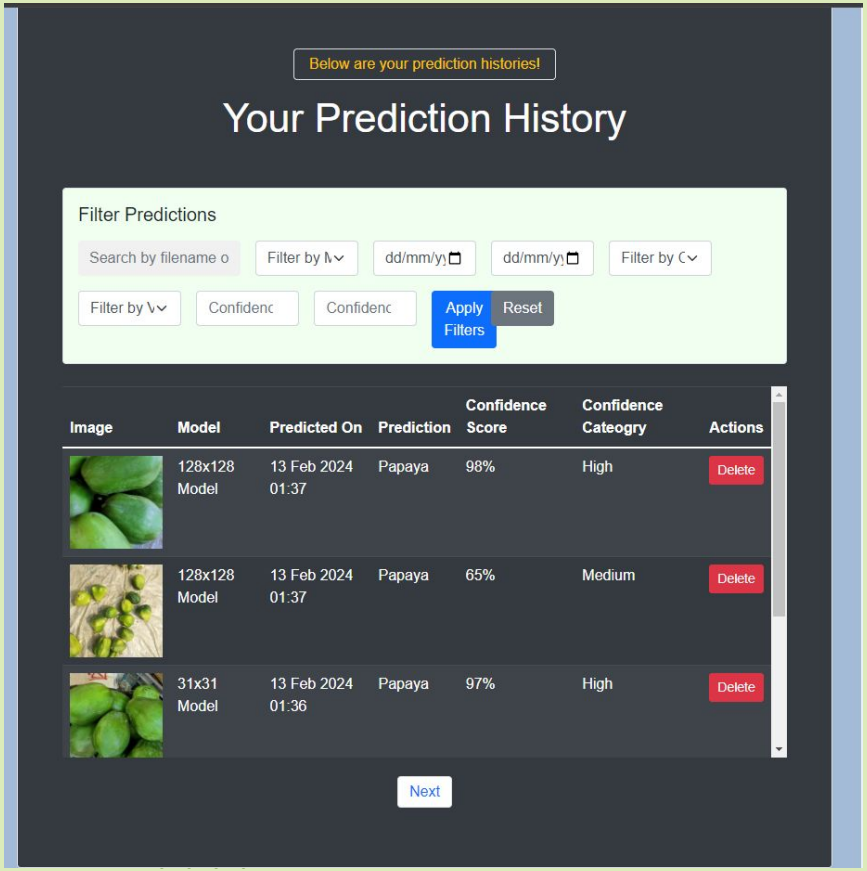





```python
# Set the login route
@app.route('/login', methods=['GET','POST'] )
def login():

    # Set up the login form here
    form = Login()

    # If methods is equal to post
    if request.method == 'POST':

        # Validate on submit here
        if form.validate_on_submit():

            # Filter the uyser by their id
            user = User.query.filter_by(email = form.email.data ).first()
            # Debugging tools
            # print(user)

            # Check whether the password is correct
            if user:
                # Utilise password hash to hash the password so that the company is not able to see
                if check_password_hash(user.password_hash , form.password.data):
                    print('login')
                    login_user(user)

                    # Redirect to predict
                    return redirect('/predict')

                # If wrong password here
                else:
                    flash('Wrong password' , 'danger')
            # If user does not exist here
            else:
                flash('User does not exist' , 'danger')

        # Unexpected error
        else:
            flash('Error, Unable to login' , 'danger')

    # Return the rendered template here
    return render_template("login.html", form=form, title="Sign Up" , index = True )
```

```python
# Set the registration methods
@app.route('/register', methods=['GET', 'POST'])
def register():

    # Set up the form for registration
    form = RegistrationForm()
    # If it is a post method
    if request.method == 'POST':
        if form.validate_on_submit():

            # Check for existing user here

            existing_user = User.query.filter_by(email=form.email

            # If there is no existing user
            if existing_user is None:

                # It sets the password and
                user = User(username=form.username.data, email=fo

                # Set the password after validation of password
                user.set_password(form.password_1.data)

                # Commit to the database
                db.session.add(user)
                db.session.commit()

                # Redirect the user to login page here
                return redirect(url_for('login'))

            # If the user already exist, warmog will be shown
            else:
                flash('A user with that email already exists.')

    # Return the render template here
    return render_template('register.html', form=form)
```

Web Page (History)



1. Prediction History was stored

Additional

1. Provide next and back option where each page only show 5 predictions
2. Providing searching and filtering function
3.

Validity Testing

Non-Api

1. Test_add_prediction_entry_directly
    a. Test entry are placed into database

Rest api test

2. Test_add_prediction_entry
    a. Test entry that are place into the database with the use of api here
3. Test_api_get_entry
    a. Test to validate getting the correct entries back
4. Test links
    a. test link whether it is able to pass the test and the correct status code
5. Test_logged_in_links
    a. Test the links after login can be access
6. test_predict_vegetable_type_big/small
    a. Test for prediction of image here
7. Test_add_user
    a. Test for adding user here
8. Test_user_login_api
    a. Test for ability to login here

Expected Failure Testing

Non-Api

1. Test_email_exist
    a. Test for duplicate emails

Rest api test

2. Test_missing_models
    a. Test for missing models provided here
3. Test_missing_values
    a. Test for missing
4. Test links
    a. test link whether it is able to pass the test and the correct status code
5. Test_get_entry_404
    a. Test whether able to get entry that are invalid
6. test_invalid credentials

Consistency Testing

Non-Api

1. Test_consistent_entry
    a. Test for consistent entry of same data

Rest api test

2. Test_prediction_consistency
    a. Test for continuous prediction here

Unexpected Failure Testing

Non-Api

1. Test_prediction_database
    a. Unexpected database connection error to test app against database failures
2. Test_query_failure
    a. Test for invalid queries

Range Testing

Non-Api

1. Test_prediction_confidence_r
    a. Test for invalid confidence score
2. Test_prediction_userid
    a. Test for invalid user id here

Some Examples

```python
'''Non-Api:
    This are the test done without the use of api
'''


#1. This is to validate those entry that are place into the database
@pytest.mark.usefixtures('authenticated_client', 'app_context')
@pytest.mark.parametrize("data", [
    {'userid': 1, 'filename': 'image1.png', 'model': 'A', 'predicted_on':
    {'userid': 2, 'filename': 'image2.png', 'model': 'B', 'predicted_on':
    {'userid': 3, 'filename': 'image3.png', 'model': 'C', 'predicted_on':
    {'userid': 1, 'filename': 'image4.png', 'model': 'A', 'predicted_on':
    {'userid': 2, 'filename': 'image5.png', 'model': 'B', 'predicted_on':
])
def test_add_prediction_entry_directly(data):
    new_prediction = Prediction(**data)

    db.session.add(new_prediction)
    db.session.commit()

    add_row = db.session.query(Prediction).filter_by(filename=data['filer

    assert add_row is not None, "Prediction should be added to the databa
    assert add_row.prediction == data['prediction'], f"The prediction sho
    assert add_row.filename ==  data['filename']
    assert add_row.userid ==  data['userid']
    assert add_row.model ==  data['model']
    assert add_row.confidence_score ==  data['confidence_score']
    assert add_row.confidence_cat ==  data['confidence_cat']

    db.session.delete(add_row)
    db.session.commit()
```

Test for validity

```python
# 1. This is to test for data that are out of the confidence score range
@pytest.mark.xfail(reason="Data out of range here for confidence score",
@pytest.mark.parametrize("out_range_cs", [
    {'userid': 1, 'filename': 'image.png', 'model': 'A', 'predicted_on':
    {'userid': 2, 'filename': 'image.png', 'model': 'A', 'predicted_on':
])
def test_prediction_confidence_r(out_range_cs):
    with pytest.raises(ValueError):
        new_prediction = Prediction(**out_range_cs)
        db.session.add(new_prediction)
        db.session.commit()
```

Test for range

```python
#2. This is to validate those entry that are place into the database with
@pytest.mark.usefixtures('authenticated_client', 'app_context')
def test_add_prediction_entry(authenticated_client):
    with authenticated_client.session_transaction() as session:
        authenticated_user_id = int(session.get('_user_id'))
    print(authenticated_user_id)
    data = {
        'filename': 'vegetable_image.png',
        'model': 'big',
        'prediction': 'Tomato',
        'confidence_score': 0.95,
        'confidence_cat':'High'
    }


    response = authenticated_client.post(f'/api/add/{authenticated_user_i
    assert response.status_code == 201, "Expected 201 Created status code
    response_data = json.loads(response.data.decode('utf-8'))
    assert 'id' in response_data, "Response data should contain an 'id' f
```

Test for validity

```python
## 6.1 Test for big too here
@pytest.mark.usefixtures("authenticated_client")
@pytest.mark.parametrize("class_name,image_path", test_data)
def test_predict_vegetable_type_big(authenticated_client, class_name, ima
    encoded_image = encode_images(image_path)

    time.sleep(3)
    data = {
        "image": encoded_image,
        "modelChoice": "big"
    }
    response = authenticated_client.post("/api/predict", json=data)
    assert response.status_code == 200, "Failed to authenticate or other
    response_body = response.json
    assert "prediction" in response_body, "Response body should include a
    assert response_body["prediction"] == class_name, f"Expected {class_n
```

Test for validity

```python
. This test is to provide unexpected database connection error to test ap
test.mark.xfail(reason="Database connection error", strict=True)
test.mark.usefixtures('authenticated_client', 'app_context')
test_prediction_database(monkeypatch):
def mock_commit(*args, **kwargs):
    raise Exception("Database connection error")

monkeypatch.setattr(db.session, "commit", mock_commit)

with pytest.raises(Exception) as exc_info:
    new_prediction = Prediction(userid=1, filename='image.png', model='/
    db.session.add(new_prediction)
    db.session.commit()

assert "Database error" in str(exc_info.value), "Unexpected failure"
```

Test for unexpected f

```python
'''
@pytest.mark.usefixtures('authenticated_client','app_context')
@pytest.mark.xfail(reason="Duplicate email", strict=True)
# 1. This is to test for data that are duplicate
def test_email_exist():
    username = f'uniqueUser{int(time.time())}'
    email = "testuser@gmail.com"

    user1 = User(username=username, email=email, joined_at=datetime.utcno
    user1.set_password('123')
    db.session.add(user1)
    db.session.commit()

    user2 = User(username=username, email=email, joined_at=datetime.utcno
    user2.set_password('123')

    db.session.add(user2)
    db.session.commit()
```

Test for expected

```python
test_data = collect_img()
@pytest.mark.usefixtures("authenticated_client")
@pytest.mark.parametrize("class_name,image_path", test_data[:1])
def test_prediction_consistency(authenticated_client, class_name, image_p
    encoded_image = encode_images(image_path)
    data = {"image": encoded_image, "modelChoice": "big"}

    for _ in range(5):
        response = authenticated_client.post("/api/predict", json=data)
        assert response.status_code == 200
        response_body = response.json
        assert "prediction" in response_body
        assert response_body["prediction"] == class_name, "Prediction sho
```

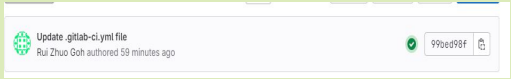
Test for consistency

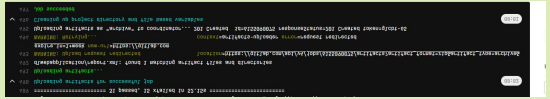
And many many more...

CICD Development

```yaml
stages:
  - test
  - build
  - deploy

variables:
  DOCKER_IMAGE_NAME: "rz111/myapp"
  DOCKER_IMAGE_TAG: "latest"

pytest:
  stage: test
  image: python:3.8
  script:
    - cd dlwebapplication
    - pip install -r requirements.txt
    - python -m pytest -s --junitxml=report.xml
  artifacts:
    when: always
    paths:
      - dlwebapplication/report.xml
    expire_in: 1 week
```

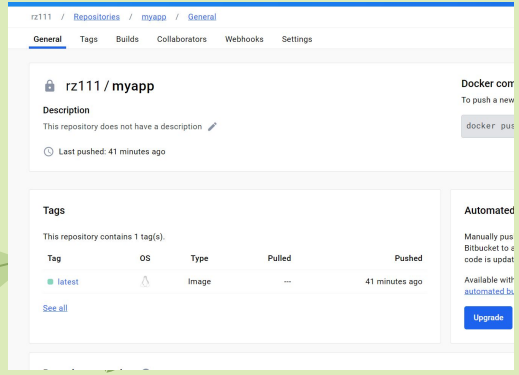
```yaml
build:
  stage: build
  image: docker:19.03.12
  services:
    - docker:dind
  before_script:
    - echo "$DOCKER_PASSWORD" | docker login -u "rz111" --password-stdin
  script:
    - docker build -t $DOCKER_IMAGE_NAME:$DOCKER_IMAGE_TAG -f dlwebapplic
    - docker push $DOCKER_IMAGE_NAME:$DOCKER_IMAGE_TAG
  only:
    - main

deployment:
  stage: deploy
  script:
    - >
      curl -X POST -H "Content-Type: application/json"
      -d "{\"serviceName\": \"myapp\", \"imageTag\": \"$DOCKER_IMAGE_TAG\
      "https://api.render.com/deploy/srv-cn47b8ocmk4c73eljmb0?key=KYlJIHy
  only:
    - main
```





Continuous Integration here with the use of pytest

Continuous Deployment here with the use of docker hub and render

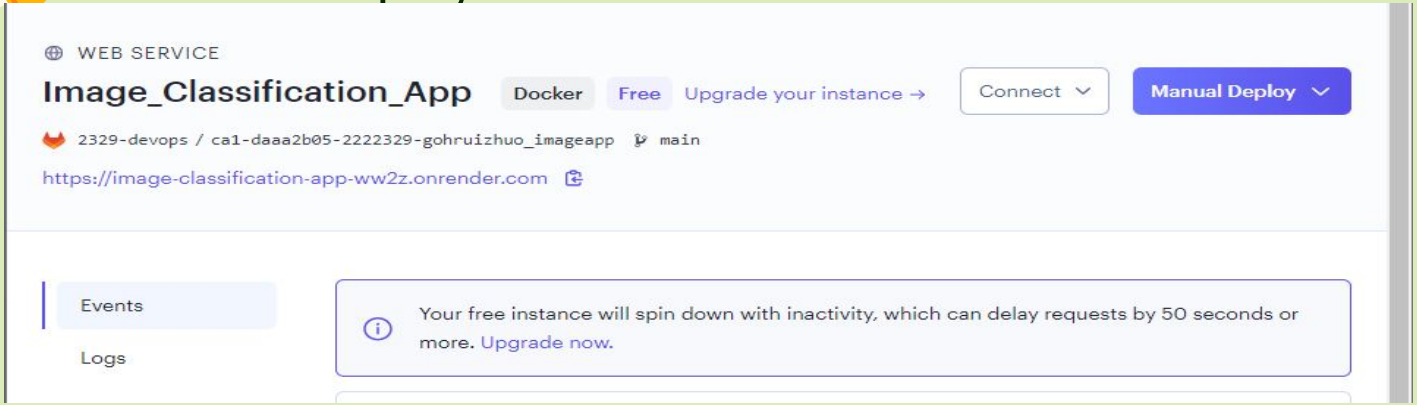


Deployed Image on docker hub

1. Provide better tracker
2. Better maintainability here with the use of docker image tag

CICD Development

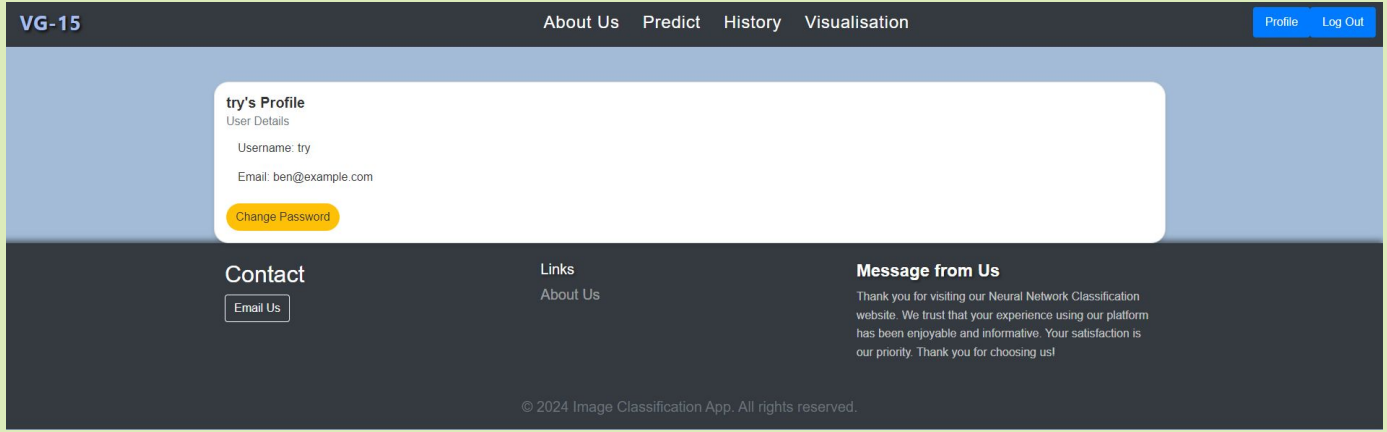
1. Deployed on render

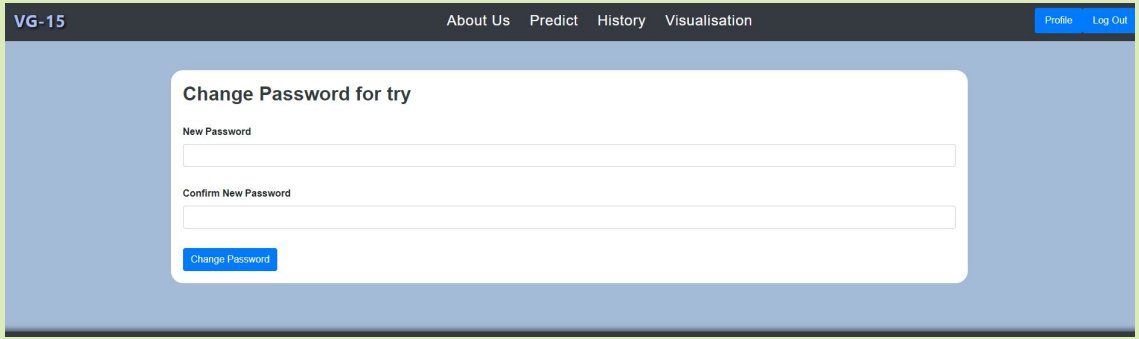


https://image-classification-app-ww2z.onrender.com
Scan to access here

Additional Features

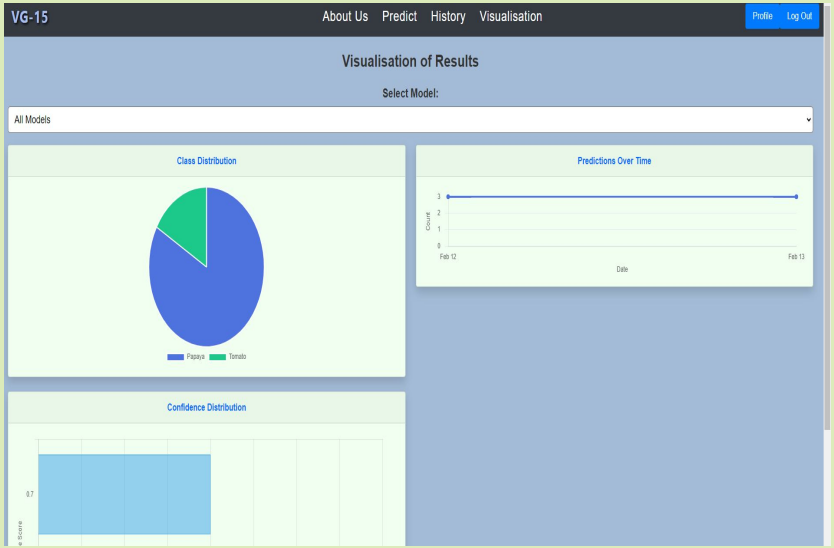
1. Created a user profile page that enables the change of password
2. Provision of change password
3. Provision of reset password

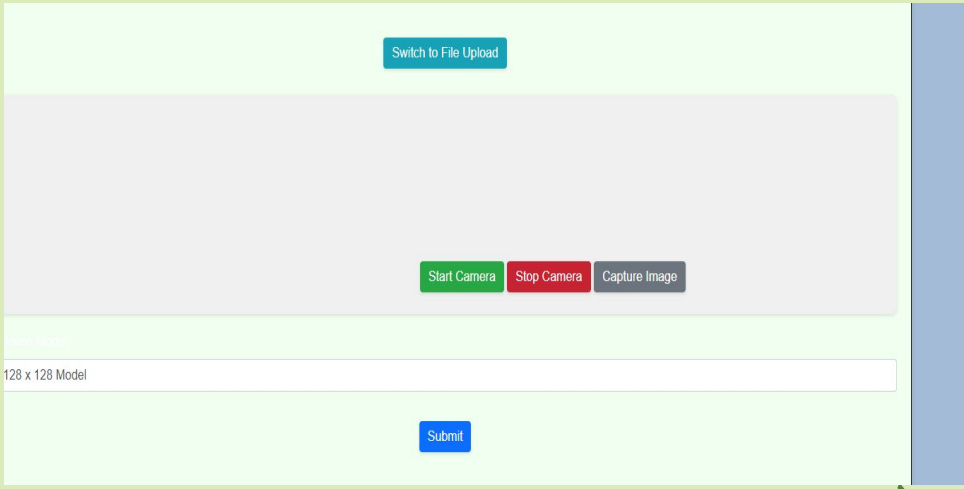




Additional Features

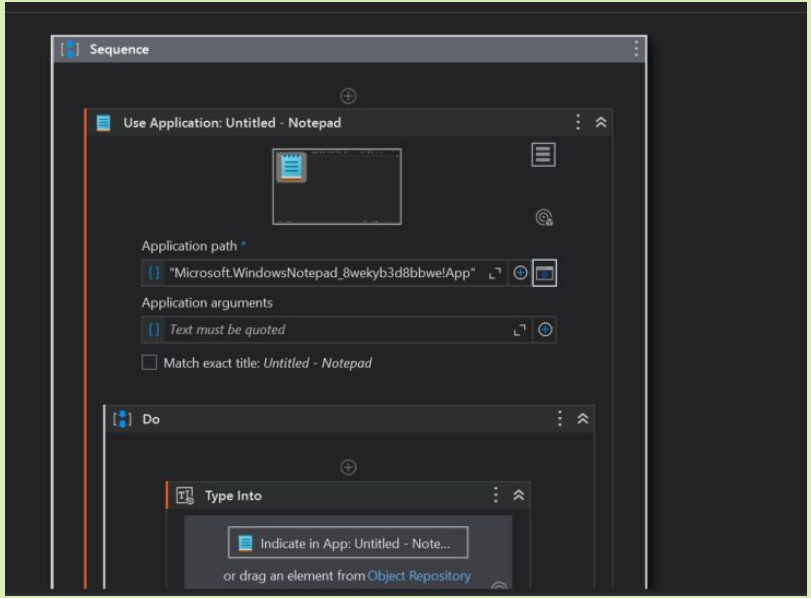
1. Provided visualisation page to visualisation results and habits here
2. Able to predict with the use of webcam here too





RPA

1. Provdie full testing automation
2. This test is from registration to login to prediction to log out
3. Below shows the workflow



RPA (Video)

THANK YOU