



PEREGRINE APP CASE STUDY

A

Case Study

Submitted in Complete Fulfillment of the Requirement for

FSE Community

Jr. FSE Engineering Program

(Direct Accreditation)

Submitted by:

SOHRAB AZAM

Employee ID: 761214

Conducted by:

Cognizant Academy

Academy & Training partner IIHT.

Technologies Implemented:

Technology Stack Java	Cloud Platform Azure	Presentation Skill Angular	Database & Storage Skill Mongo DB	Compute & Integration Skill Kafka
---------------------------------	--------------------------------	--------------------------------------	---	---

ACKNOWLEDGEMENT

It is a great pleasure for me to express my earnest gratitude to **all the mentors at IIHT, R, Jaisree** (Batch Owner at Cognizant Academy), and all those who have been associated with the program. I am very much grateful to **IIHT** for the encouragement, valuable suggestions, innovative ideas, and supervision throughout this project work, without which the completion of the project work would have been a difficult one and also thank Cognizant Academy for giving me this fantastic opportunity.

I would also like to express my thanks to my Project Manager, **Garima Agrawal** for encouraging and motivating me to enroll and take up CDE practices. Also, I would like to thank my Supervisor **I P, Ramyadevi** for creating the necessary environment which made it easier to cope up with both the project deliverables and this case study as well.

My sincere thanks to all the mentors of the IIHT department for their precious help, encouragement, kind cooperation and suggestions throughout the development of the project work.

Lastly, I would like to express my gratitude to Cognizant for providing me with a congenial working environment and for this lovely opportunity.

Sohrab Azam

AIMS OF THIS CASE STUDY

The aims of the project are listed below:

- To create an analysis of a Tweet App, which is an SPA (Single Page App) for registered users to post new tweets, reply to tweets, like or unlike tweets.
- This document also serves to establish the traceability between the Business Objectives and the requirements identified in the proposed solution and how they satisfy the stated objectives.
- Provide expectation traceability in terms of the requirements and the user expectation.
- Serves as a formal template for documenting the Business Requirements which also includes statutory and regulatory requirements.
- The purpose of this document is to systematically capture requirements for the project and the system to be developed. Functional requirements are captured in this document. It also serves as the input for the project scoping.

APPLICATION OVERVIEW

Below are the **summarized objectives** for the fulfillment of this project:

1. Tweet app contain the following core modules:
 - User Registration and Login
 - Dashboard where all the tweets posted by registered users will be displayed.
 - Tweet body will have all the tweet information and will be associated with a comment section
 - Profile Page for posting a tweet and personal details section
 - User handle for viewing users and their tweets
2. Whenever somebody registers, their encrypted credentials are to be stored in a database and validated while they register. They can login only if they are already registered.
3. Logged-in users can post tweets, can edit and delete only their own tweets and comments and like/unlike their own and others comments.

Following technologies implemented for the completion of the application:

Angular	Karma & Jasmine	Java	Gradle	Mockito
Junit	Spring Boot	Kafka	MongoDB	Docker
Sonarqube	Azure App Services	Azure Container Registry	Azure Notification Hubs	Azure DevOps

DETAILED PROJECT DESCRIPTION

FEATURES OF THE APPLICATION	
REGISTRATION	<ul style="list-style-type: none">• A user is able to register in the tweet application.• User will not be able to register with the same email and username more than once (unique email and username).• Validations have been implemented in the front-end and back-end for the registration.• In the registration page, if all field validations are passed only then the signup button will appear.• Success or failure messages for validation, signup and server error will be displayed on the basis of response from the backend.• The user is logged in automatically after successful registration using existing logic. Post successful registration, the user will land on the dashboard page.
LOGIN	<ul style="list-style-type: none">• Only registered user is able to login to the tweet application using username and password.• Relevant messages are displayed for username does not exist, or invalid credentials.• A user is logged in to the application with a jwt token within the header response and stored in cookies. The jwt is valid only for a specific user for a given timeframe and cannot be used to hit rest client for other users and outside the timeframe.• The jwt is valid for until a day, while the user will be logged off or will not be able to use the backend services using the expired jwt.• Post successful login, the user will land on the dashboard page.
DASHBOARD PAGE	<ul style="list-style-type: none">• The dashboard shows tweets posted by all the users.• The latest tweets appear at the top.

	<ul style="list-style-type: none">• There is a refresh button and a profile button.• The refresh button refreshes the payload of the tweets and loads in fresh tweets at the top.• The profile button will help navigate to the profile page of the user• Here users can like/unlike any tweet, while edit and delete their own tweets only.• Users can comment, like/unlike any comment, while edit and delete only their own comments.• The number of heart likes will be displayed for all tweets and comments.• A logout button appears on the top-right corner of the screen which will help the user to log out of the application. The jwt is cleared from the front-end.
PROFILE PAGE	<ul style="list-style-type: none">• It consists of all the tweets from the logged in user where the user can add new tweet, delete existing tweet, edit a tweet, like, remove like, comment, edit comment, remove comment, like comment.• Profile consists of two highlighted buttons, Dashboard, which takes to dashboard page and Details. Details shows the details of the user, such as email id and contact as well as gives an option to change the password of the user.

APACHE KAFKA

Apache Kafka is a framework implementation of a software bus using stream-processing. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. publish-subscribe messaging system allows a sender to send/write the message and a receiver to read that message. In Apache Kafka, a sender is known as a **producer** who publishes messages, and a receiver is known as a **consumer** who consumes that message by subscribing it.

Here, the kafka producers and consumers are written within the spring boot application. There are **two topics** created in this project for streaming events into the kafka stream:



1. **'tweet-event'**: For events associated to tweet operations.
2. **'comment-event'**: For events associated to comment operations.

There are two producers which produces two types of event objects respectively, namely, TweetEvent and CommentEvent.

<u>TWEET EVENT</u>	<u>COMMENT EVENT</u>
Integer <code>tweetEventId</code> ; TweetEventType <code>tweetEventType</code> ; String <code>currentUser</code> ; Tweet <code>tweet</code> ;	Integer <code>commentEventId</code> ; CommentEventType <code>commentEventType</code> ; String <code>currentUser</code> ; Comment <code>comment</code> ;

For each event we have respective producers and consumers. Producers will produce the events and push it to the Kafka Stream. The Consumers will consumer the message from respective topics and operate on the events and store it in database. After consuming the message is acknowledged.

TweetEvent Object details:

1. **tweetEventId**: The id with which the tweet event is posted to the topic
2. **tweetEventType**: An enum for holding the type of tweet operation.
NEW | UPDATE_MESSAGE | UPDATE_LIKE | DELETE
3. **currentUser**: The logged-in user who posted the tweet event
4. **tweet**: Holds the data for the tweet.

CommentEvent Object details:

1. **commentEventId**: The id with which the tweet event is posted to the topic
2. **commentEventType**: An enum for holding the type of tweet operation.
NEW | UPDATE_MESSAGE | UPDATE_LIKE | DELETE
3. **currentUser**: The logged-in user who posted the comment comment
4. **comment**: Holds the data for the comment.

Note: Event does not mean just tweet or comment is posted. It means a tweet or comment operation is being performed and pushed to the kafka streaming hub. The operations include new post, updating tweet/comment, like/unlike and delete operations on a tweet/comment.

```
GRADLE IMPORTS  
implementation 'org.springframework.kafka:spring-kafka:2.8.1'  
testImplementation 'org.springframework.kafka:spring-kafka-test:2.8.1'
```



RUNNING KAFKA IN LOCAL:

1. Download Kafka,set it in a drive, for example D drive and open cmd
2. Goto config folder copy server.properties and create server-1.properties and server-2.properties
3. Edit server-1.properties
broker.id=1 (change value to 2 in server-2.properties)
auto.create.topics.enable=false
log.dirs=/tmp/kafka-logs-1 (change value to 2 in server-2.properties)
listeners=PLAINTEXT://localhost:9093 (change value to 9094 in server-2.properties)
4. Type:

D:\kafka\kafka_2.12-2.8.1\bin\windows\zookeeper-server-start.bat

D:\kafka\kafka_2.12-2.8.1\config\zookeeper.properties

D:\kafka\kafka_2.12-2.8.1\bin\windows\kafka-server-start D:\kafka\kafka_2.12-2.8.1\config\server.properties

D:\kafka\kafka_2.12-2.8.1\bin\windows\kafka-server-start D:\kafka\kafka_2.12-2.8.1\config\server-1.properties

D:\kafka\kafka_2.12-2.8.1\bin\windows\kafka-server-start D:\kafka\kafka_2.12-2.8.1\config\server-2.properties

Zookeeper and 3 brokers will be set up in the local. Following is the snippet from application.yml:

```
producer:
  bootstrap-servers: localhost:9092,localhost:9093,localhost:9094
  key-serializer: org.apache.kafka.common.serialization.IntegerSerializer
  value-serializer: org.apache.kafka.common.serialization.StringSerializer
consumer:
  bootstrap-servers: localhost:9092,localhost:9093,localhost:9094
  key-deserializer:
    org.apache.kafka.common.serialization.IntegerDeserializer
  value-deserializer:
    org.apache.kafka.common.serialization.StringDeserializer

group-id: tweet-events-listener-group
```

MONGO DB ATLAS

MongoDB Atlas is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS, Azure, and GCP). MongoDB Atlas is the best way to deploy, run, and scale MongoDB in the cloud.

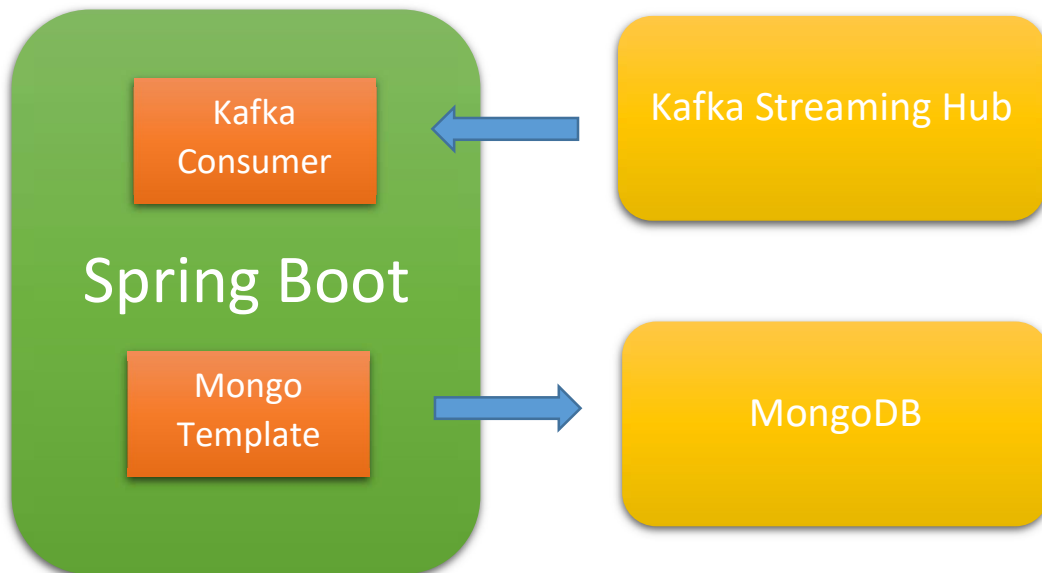
For this project have set up an account in MongoDB Atlas.

GRADLE IMPORTS

```
implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
```

In `application.yml`:

```
spring.data.mongodb.uri=<Connection string value from mongodb atlas>  
spring.data.mongodb.database=spring
```



SAMPLE USER ACCOUNT DATA IN `useraccount`

```
_id: ObjectId("61ffc7a7dc347d60d3475e4d")
fname: "John"
lname: "Doe"
email: "john.doe@email.com"
password: "e94256dd961f4292f0acfd68558740fb253afc4e8af7f098fa4e9d5f206efa05"
phone_number: "1234567890"
createDttm: 2022-02-06T13:05:43.545+00:00
updateDttm: 2022-02-06T13:05:43.545+00:00
username: "JohnDoe1234"
avatar: "man"
_class: "com.peregrine.register.bean.PeregrineUser"
```

SAMPLE TWEET DATA IN `tweetData`

```
_id: ObjectId("620122b810989a4be23ab76b")
tweetId: "sohrabazam123-bb922236-2b73-4b05-b8bd-6972f086b22a"
message: "hello"
createDttm: 2022-02-07T13:46:32.420+00:00
updateDttm: 2022-02-07T13:46:32.420+00:00
username: "sohrabazam123"
likes: 0
> likesUsernames: Array
  noOfComments: 0
_class: "com.peregrine.tweet.bean.TweetBean"
```

SAMPLE COMMENT DATA IN `commentData`

```
_id: ObjectId("6202d4d24c96b74bd5296606")
commentId: "sohrabazam123-77319307-912c-44d7-9042-f5a158f676e9"
message: "Message2"
createDttm: 2022-02-08T20:38:42.004+00:00
updateDttm: 2022-02-08T20:38:42.004+00:00
username: "sohrabazam123"
tweetId: "steve12345-b758dc28-fdfe-49b2-a0bd-5ee76f89af31"
likes: 0
> likesUsernames: Array
_class: "com.peregrine.tweet.bean.CommentBean"
```

SPRING BOOT

Spring Boot is an open-source Java-based framework used to create a micro-Service. Name of the application Peregrine App. Peregrine name comes from the fastest bird in the world, which is the peregrine falcon.

Controllers:

There are four controllers:

1. Login Controller
2. Registration Controller
3. Tweet Controller
4. Comment Controller

REGISTRATION REST ENDPOINTS:

SR. NO.	DESCRIPTION	METHOD	URL
1	Check Email	POST	/api/v1.0/tweets/email
2	Check username	POST	/api/v1.0/tweets/username
3	Register a user	POST	/api/v1.0/tweets/register

LOGIN REST ENDPOINTS:

SR. NO.	DESCRIPTION	METHOD	URL
1	Reset Password	PUT	/api/v1.0/tweets/resetPassword
2	Login User	POST	/api/v1.0/tweets/email

TWEET CONTROLLER REST ENDPOINTS:

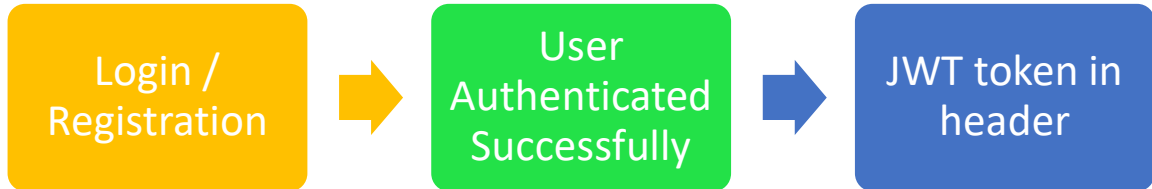
SR. NO.	DESCRIPTION	METHOD	URL
1	Post a tweet	POST	/api/v1.0/tweets/postTweet
2	Retrieve all tweets of a user	POST	/api/v1.0/tweets/fetchTweets/username
3	Retrieve all tweets	GET	/api/v1.0/tweets/fetchTweets
4	Update a tweet	PUT	api/v1.0/tweets/updateTweet
5	Like / Dislike a tweet	PUT	api/v1.0/tweets/likeTweet
6	Delete a tweet	DELETE	api/v1.0/tweets/deleteTweet

COMMENT CONTROLLER REST ENDPOINTS:

SR. NO.	DESCRIPTION	METHOD	URL
1	Post a comment	POST	/api/v1.0/tweets/postComment
2	Retrieve all comments of a tweet	POST	/api/v1.0/tweets/fetchComments/tweetId
3	Update a comment	PUT	api/v1.0/tweets/updateComment
4	Like / Dislike a comment	PUT	api/v1.0/tweets/likeComment
5	Delete a comment	DELETE	api/v1.0/tweets/deleteComment

JWT AUTHORIZATIONS:

JWT token is returned in the header response if successfully authenticated.



```
authorization → Bearer  
eyJhbGciOiJIUzUxMiJ9.eyJzdWUiOiJzZGV2ZXJvZ2VycyEyMzQ1IiwiaXhwaWJjoxNjQ1MzgxMDUyLCJpYXQiOiJ  
E2NDQ3NzYyNTJ9.ix3UFk1xWlyuuJdFtb_z-
```

JWT RULES:

1. Jwt token generated is valid for specific users and for specific timeframe (expiry of the token is set to 1 day). Post expiry the jwt token will get invalidated.
2. User can post new tweet, comment on a tweet, like / unlike comment or tweet, edit comment or tweet, delete comment or tweet using their own jwt token. Jwt token of another user cannot be used to post, like, update or delete comment or tweet of another user.
3. Jwt is not require to at the time of login, registration, check username or email exists service calls.
4. Jwt is stored in the front end using a cookie in encrypted format.

```
String[] allowDomain = {"http://localhost:4200",  
                        "http://localhost:8080",  
                        "https://peregrineapp.azurewebsites.net"};
```

JWT USAGE TO HIT REST CLIENT:

In the request headers, pass as Authorization as we receive in the response headers of login:

```
authorization → Bearer  
eyJhbGciOiJIUzUxMiJ9.eyJzdWUiOiJzZGV2ZXJvZ2VycyEyMzQ1IiwiaXhwaWJjoxNjQ1MzgxMDUyLCJpYXQiOiJ  
E2NDQ3NzYyNTJ9.ix3UFk1xWlyuuJdFtb_z-
```

MODELS:

TWEET

```
private String tweetId;  
private String message;  
private Date createDttm;  
private Date updateDttm;  
private String username;  
private Integer likes;  
private String likeByUser;  
private List<String> likeUserNames;  
private String avatar;  
private Long noOfComments;
```

TWEET REQUEST

```
private String id;  
private String tweetId;  
private String message;  
private String username;  
private Integer likes;  
private String likeUsername;
```

COMMENT

```
private String commentId;  
private String message;  
private Date createDttm;  
private Date updateDttm;  
private String username;  
private String tweetId;  
private Integer likes;  
private String likeByUser;  
private List<String> likeUserNames;
```

COMMENT REQUEST

```
private String commentId;  
private String message;  
private Date createDttm;  
private Date updateDttm;  
private String username;  
private String tweetId;  
private Integer likes;  
private String likeUsername;
```

TWEET RESPONSE

```
private String username;  
private List<SvcMessage> messages;  
private List<Tweet> tweets;  
private Tweet tweet;
```

COMMENT RESPONSE

```
private String tweetId;  
private List<SvcMessage> messages;  
private List<Comment> comments;
```

SVC MESSAGE

```
private String code;  
private String message;  
private String type;
```

RESPONSE MESSAGES AND THEIR CODES:

#Registration Errors

REG_W_1001=Email should not be empty.
REG_W_1002=Please enter a valid email id.
REG_W_1003=Password should not be empty.
REG_W_1004=Password length is less than the minimum length.
REG_W_1005=Password length is greater than the maximum length.
REG_W_1006=Password must have special characters.
REG_W_1007=First name should not be empty.
REG_W_1008=Last name should not be empty.
REG_W_1009=User name should not be empty.
REG_W_1010=Please enter a valid first name.
REG_W_1011=Please enter a valid last name.
REG_W_1012=Please enter a valid user name.
REG_W_1013=Please enter a valid phone number.
REG_I_1014=User registered successfully.

#Login Errors

LOG_W_1014=Invalid credentials.
LOG_W_1015=User name does not exists.
LOG_I_1017=User authenticated successfully.

#Tweet Errors

TWT_W_1016=No tweet in topic for the event.
TWT_I_1001=Tweet posted successfully.
TWT_W_1002=Tweet posting failed.
TWT_I_1003=Tweets retrieved successfully.
TWT_E_1004=Error in retrieving tweets.
TWT_I_1005=Tweet updated successfully.
TWT_E_1006=Error in updating tweet.
TWT_I_1007=Tweet deleted successfully.
TWT_E_1008=Error in deleting tweet.
TWT_W_1009=Cannot insert or update tweet of another user.

CMT_I_1001=Comment posted successfully.
CMT_W_1002=Comment posting failed.
CMT_I_1003=Comments retrieved successfully.
CMT_E_1004=Error in retrieving comments.
CMT_I_1005=Comment updated successfully.
CMT_E_1006=Error in updating comment.
CMT_I_1007=Comment deleted successfully.
CMT_E_1008=Error in deleting comment.
CMT_W_1009=Cannot insert or update comment of another user.

LOG_I_1020=Password changed successfully.
LOG_W_1021=Cannot change password of another user.
LOG_E_1022=Error in changing the password.

#Common Errors

APP_E_SVC_UNAVAILABLE=Service Unavailable.

ANGULAR

Angular-13: Angular is a very popular front-end framework for developing websites and is used by many organizations. Hereby in Peregrine also Angular is used.

Angular Material: An inhouse components library which is compatible with Angular which provides various components such as cards, accordions, dropdowns, buttons, icons etc for ready to use with theme features enabled.

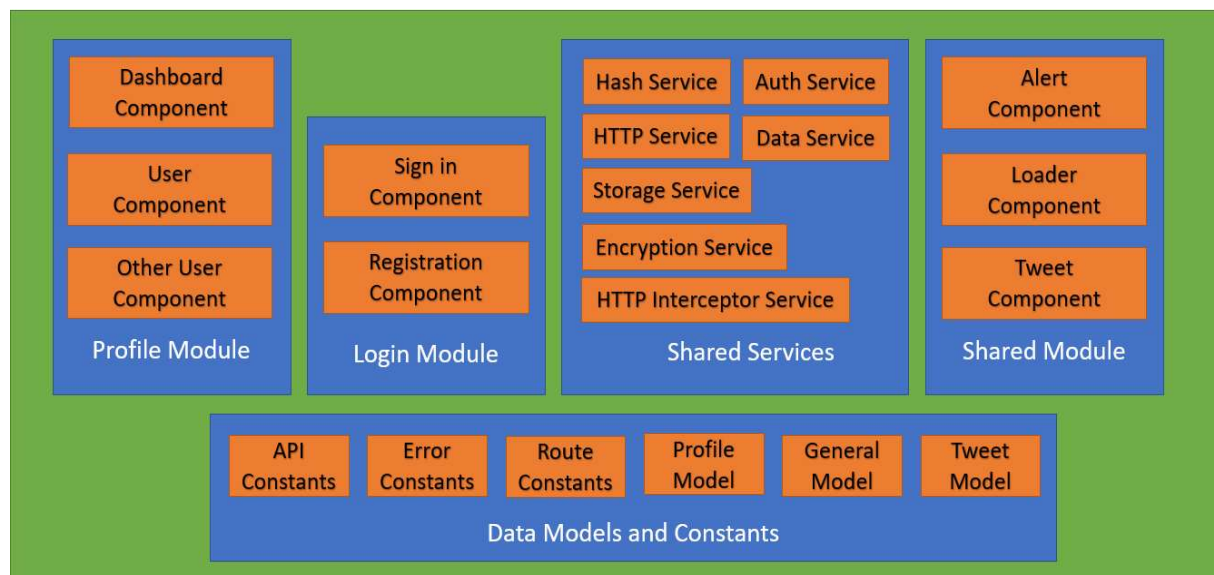
Bootstrap: Another library for frontend development which is mainly used for its handy grid layout and elements arrangement feature as well as responsiveness of the website.

Ngx-cookie-service: Used for cookie storage and retrieval, this library comes in handy and easy to use functions.

Crypto-js: Used for encryption of any string value with a given key, a popular library for the same

Sha.js: A hashing library based on the type of hashing can hash a given string very easily with easy-to-use functions.

FRONT-END COMPONENT AND MODULES

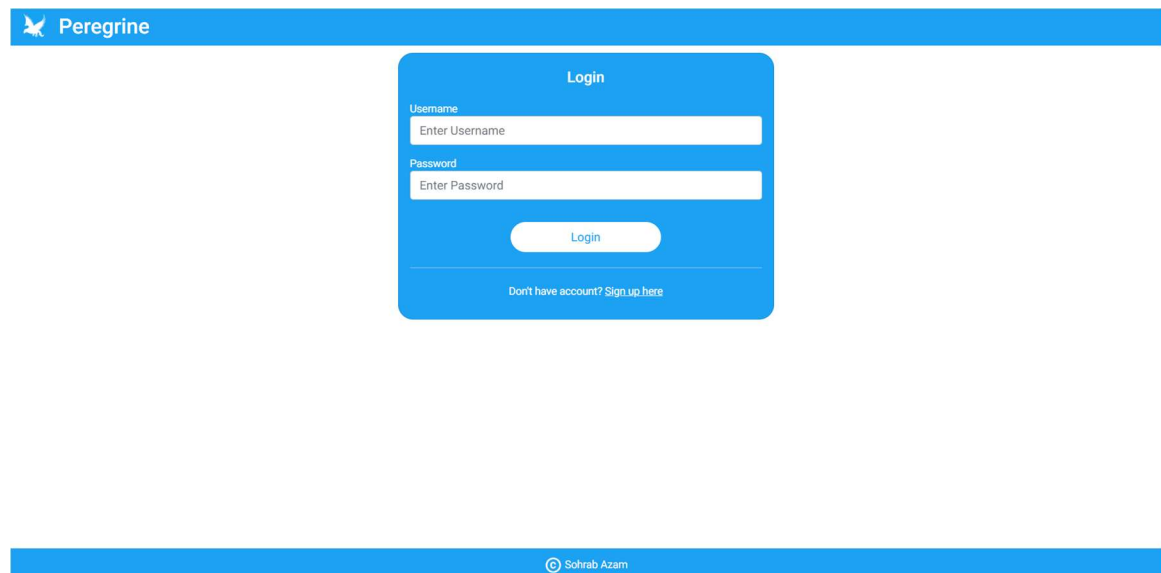


URL

<https://peregrineapp.azurewebsites.net>

LOGIN PAGE:

Opening the website <https://peregrineapp.azurewebsites.net>, user lands on login page under the URL <https://peregrineapp.azurewebsites.net/login>, hereby an existing user can login or a new user can click on Sign-up here button for navigating to signup page.



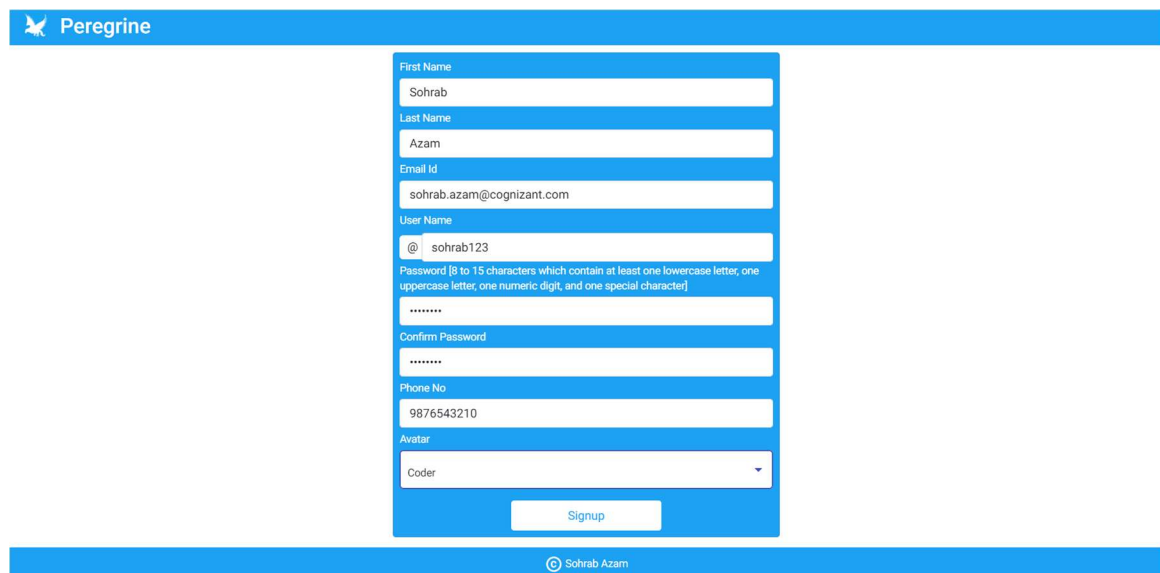
The screenshot displays the login interface of the Peregrine application. At the top, a blue header bar features the Peregrine logo (a bird icon) and the brand name "Peregrine". The main content area is white and contains a blue login card. The card is titled "Login" and includes two input fields: "Username" with the placeholder text "Enter Username" and "Password" with the placeholder text "Enter Password". Below these fields is a blue "Login" button. At the bottom of the card, there is a link that reads "Don't have account? Sign up here". The footer of the page is a blue bar containing the copyright notice "© Sohrab Azam".

REGISTRATION PAGE:

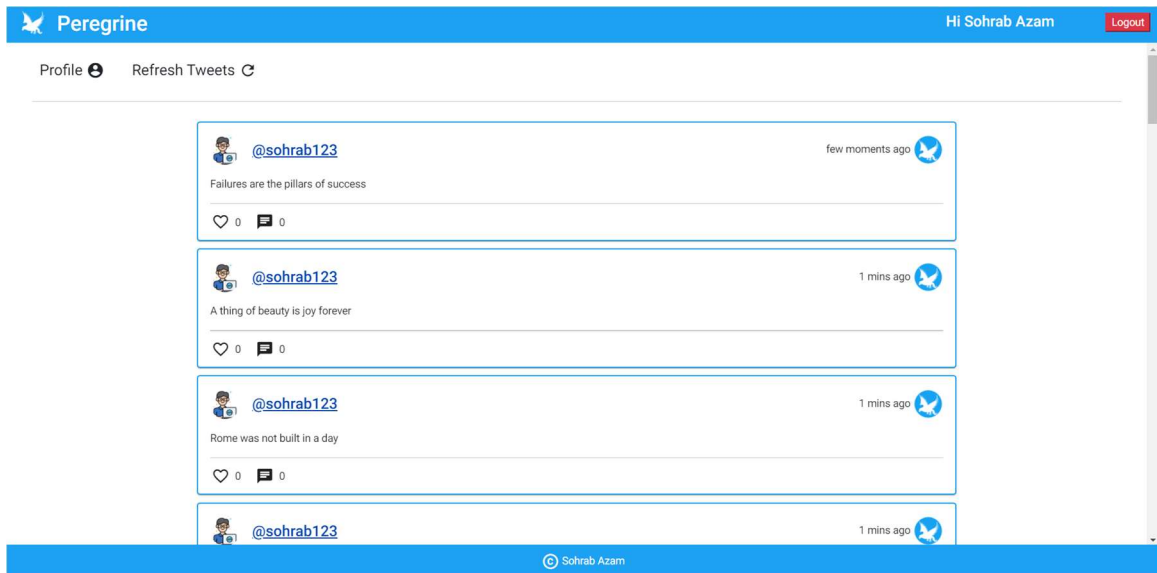
Upon navigation to signup page <https://peregrineapp.azurewebsites.net/login/register>, the user can enter the details as per the following form fields:

- First Name
- Last Name
- Email id
- Username (a combination of alphanumeric and number)
- Password (8-15 characters with a combination of at least 1 uppercase, 1 lowercase, 1 number and 1 special character)
- Confirm Password
- Phone number (10 digits only)
- An Avatar from the dropdown menu.

Following the form fill up user will be able to see the Signup button instead of Login, until then user can go back to login page using the same button. If the form is filled with incorrect data, then a red highlight appears on the box and then when the data is correct the highlight turns gets removed. On click of Signup the user is registered onto the application and taken to the dashboard page.



The screenshot displays the registration interface for the 'Peregrine' application. At the top, a blue header bar features the 'Peregrine' logo and name. The registration form is a vertical stack of input fields with blue borders, set against a white background. The fields are: 'First Name' (containing 'Sohrab'), 'Last Name' (containing 'Azam'), 'Email Id' (containing 'sohrab.azam@cognizant.com'), 'User Name' (containing '@ sohrab123'), 'Password' (with a placeholder text: 'Password [8 to 15 characters which contain at least one lowercase letter, one uppercase letter, one numeric digit, and one special character]' and masked with dots), 'Confirm Password' (masked with dots), 'Phone No' (containing '9876543210'), and 'Avatar' (a dropdown menu showing 'Coder'). A 'Signup' button is positioned at the bottom of the form. The footer consists of a solid blue bar with the text '© Sohrab Azam' on the right.

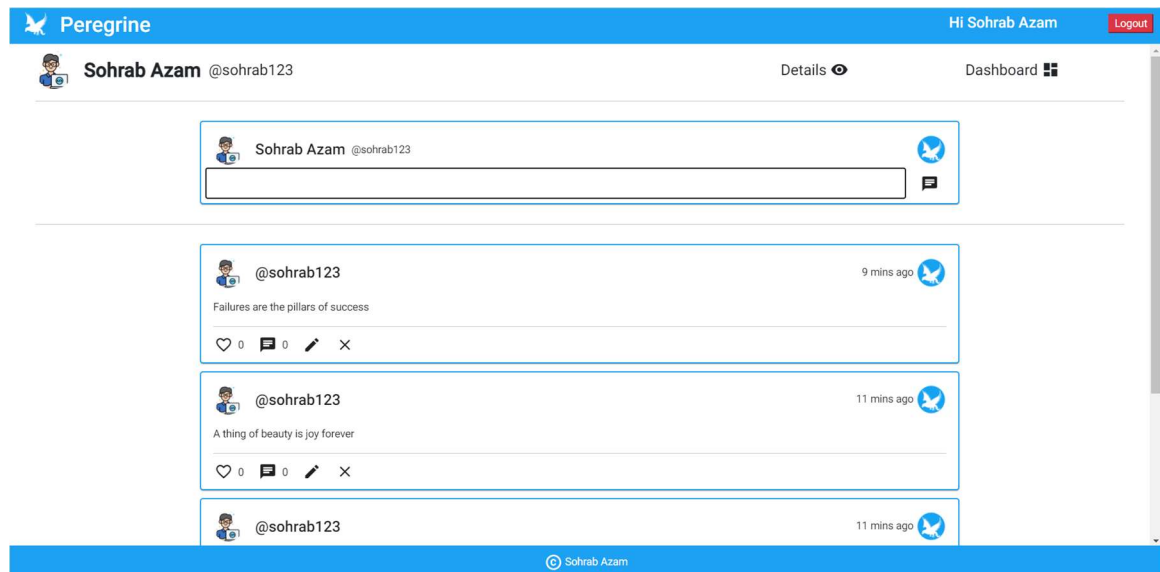


DASHBOARD PAGE:

1. Dashboard page, <https://peregrineapp.azurewebsites.net/profile>. Following are the features of the dashboard:
2. Tweet from all users across the application will appear in this page.
3. A logged in user can comment, like, unlike (if liked before, same as removing the like) on Tweets posted by self or other users.
4. The comments can also be liked. If the comment is made from the logged in user, the user can delete or edit the comment.
5. Dashboard consists of 3 flows, Goto Profile, Refresh Tweets and Username links.
Goto Profile takes to profile page of the user.
Refresh Tweets refreshes the tweets and loads the latest tweets.
Username handle link takes to a page where all the tweets by that particular user clicked is shown at one place.

PROFILE PAGE:


1. Profile page, <https://peregrineapp.azurewebsites.net/profile/user>, It consists of the following features:
2. Tweets from the logged in user where he can add new tweet, delete existing tweet, edit a tweet, like, remove like, comment, edit comment, remove comment, like comment.
3. Profile consists of two highlighted buttons:
 - Dashboard, which takes to dashboard page
 - Details. Details shows the details of the user, such as email id and contact as well as gives an option to change the password of the user.
4. Other User page consists of tweets from a selected user from the dashboard page. It is same as dashboard page with the filter of tweets from a particular user. There is one button here for navigating to dashboard, the Dashboard button.



DETAILS SECTION:

Peregrine


Hi Sohrab AzamLogout


 **Sohrab Azam** @sohrab123

Details

Dashboard





Email : sohrab.azam@cognizant.com Contact : 9876543210 Password : *****


 **Sohrab Azam** @sohrab123

 @sohrab123

9 mins ago





Failures are the pillars of success


 0  0  0 

 @sohrab123

11 mins ago

A thing of beauty is joy forever


 0  0  0 

 Sohrab Azam


OTHER USER'S PAGE:

Peregrine

Hi Sohrab AzamLogout



 **@JohnDoe1234**


Dashboard

 @JohnDoe1234

11 hours ago



test


 1  1

 @JohnDoe1234

1 weeks ago

asdad

 0  0

 Sohrab Azam

DATA FLOW

The user registers then the data is sent to the backend via the **Http Service**, which handles all the http requests outgoing and incoming to the application, it uses HTTP Client Module from Angular for its purpose. The Http Service is capable of handling Headers and creating observe: response option for the http response handling. It consists of four methods **get, post, put** and **delete**. The methods are used for the respective REST Service calls. The user data is sent to backend which is handled by the backend of the application and a response is sent upon successful addition of user to database. The **JWT** sent from the backend is stored in the **Data Service's jwt** variable which has a getter, setter and an observable for data access for use throughout the application and for sending it back along with further requests for verification the logged in user. Then **loginState** variable in Data Service is set to true to indicate the user has logged in onto the application and the data is used to activate the routes to various pages which is guarded by **Can Activate** route guard from angular auth guard in the **Auth Service**. The user data is also stored in **profile** variable which has a getter, setter and observable attached to it.

The Data Service stores the data of loginState, profile and JWT after encryption using the **Encryption Service** and **Cookie Storage Service** in the cookie for ease of refresh and reload of the page.

Hash Service is used to hash the password before sending to backend as a measure of security. It's done both at time of registration and login

HTTP Interceptor Service is responsible for intercepting http response which can detect any errors and redirect to login page for error 401 and 403.

The data in dashboard is fetched from backend using the HTTP Service which consists of tweets of all users which is shown. On click of comment the comments are fetched for a particular tweet from the backend.

The data in profile and other user page is fetched from backend which is tweets pertaining to a particular user only. On click of comment the comments for a particular tweet is fetched and shown.

Post tweet and add comment data is added to the database instantly as its posted. Edit, like, remove like, delete is also real-time.

Upon logout add data from Data Service is cleared which includes the Cookies as well.

Tweet Component is responsible for showing, adding and handling tweet data, requests and responses across various pages.

Alert Component is responsible for showing alerts on the application based on requirement. It has various categories success, info, warning, danger and neutral. Mainly danger, warning and success is used across the application

Loader Component shows the loader on the application indicating a wait time for the user.

DOCKER

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. This application makes use of 2 docker containers: one for frontend and another for backend.

ANGULAR APPLICATION DOCKERFILE

```
FROM node:14.17.0
ENV NODE_ENV=/peregrineFE/node_modules

# Create a directory where our app will be placed
RUN mkdir -p /peregrineFE/

# Change directory so that our commands run inside this new directory
WORKDIR /peregrineFE/

# Get all the code needed to run the app
COPY . /peregrineFE/

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD [ "npm", "run", "start:docker:fe" ]
```

SPRING BOOT APPLICATION DOCKERFILE

```
# GRADLE BUILD
FROM gradle:7.3.3-jdk11-alpine AS build-image
WORKDIR /home/app
COPY . /home/app
RUN gradle --no-daemon build

# SPRING BOOT
FROM openjdk:11.0.9.1-jre
VOLUME /tmp
EXPOSE 8080
COPY --from=build-image /home/app/build/libs/PeregrineApp-0.0.1-SNAPSHOT.jar
app.jar
ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -
Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
```

SOME DOCKER COMMANDS:

BUILDING AN IMAGE AND RUNNING IT ON LOCAL

```
docker build -t peregrinecontainers.azurecr.io/peregrine-app
```

```
docker run -p 8080:8080 peregrinecontainers.azurecr.io/peregrine-app
```

PUSHING IMAGE DIRECTLY INTO THE AZURE REGISTRY

```
docker login peregrinecontainers.azurecr.io
```

```
docker tag peregrineapp-release1_peregrine-app  
peregrinecontainers.azurecr.io/peregrine-app
```

```
docker push peregrinecontainers.azurecr.io/peregrine-app
```

KARMA AND JASMINE



SONARQUBE

SonarQube is an open-source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells on 20+ programming languages.

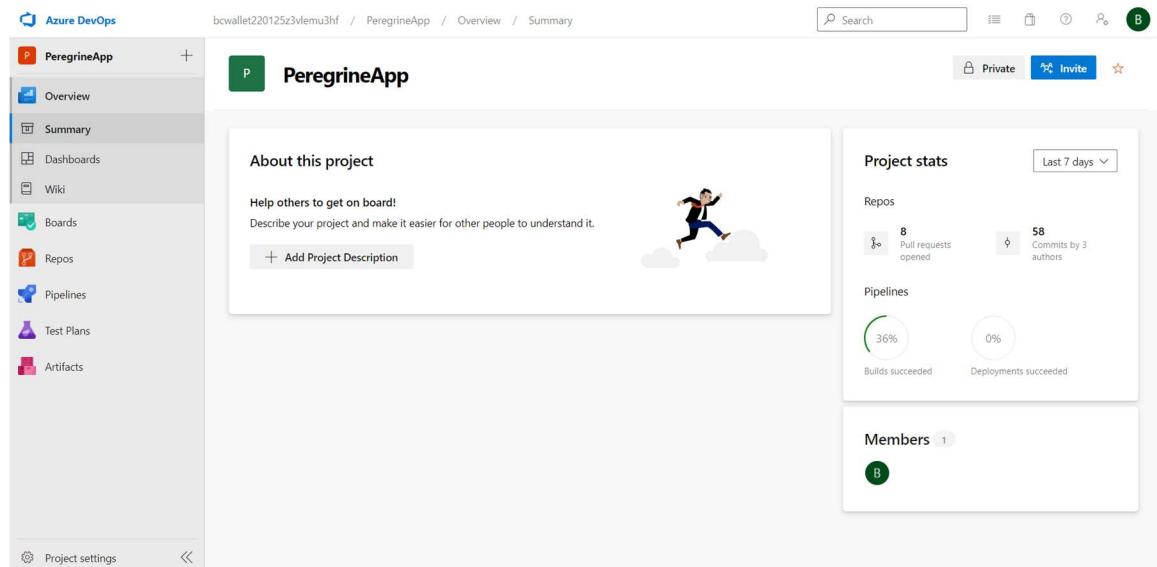
```
gradle sonarqube -Dsonar.projectKey=PeregrineApp -Dsonar.host.url=http://localhost:9000 -Dsonar.login=c0e8ecce197fb654904dc321303eb3996468d67b
```

```
sonarqube {
    properties {
        property 'sonar.host.url', 'http://localhost:9000'
        property "sonar.coverage.exclusions", "***-
/com/peregrine/constant/*','**-/com/peregrine/kafka/controller/*'"
    }
}
tasks.named('sonarqube').configure {
    dependsOn test
}
jacocoTestReport {
    reports { xml.enabled true
    }
}
test {
    useJUnitPlatform()
    forkEvery=1
    finalizedBy jacocoTestReport
}
```

The screenshot shows the SonarQube web interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar is present on the right. The main content area displays the project 'PeregrineApp' with a 'Failed' status. The project details include a table of metrics: Bugs (26), Vulnerabilities (0), Hotspots Reviewed (20.0%), Code Smells (179), Coverage (42.0%), Duplications (5.0%), and Lines (2.9k). The left sidebar contains filters for Quality Gate, Reliability, Security, and Security Review. A warning message at the bottom states: 'Embedded database should be used for evaluation purposes only'.

AZURE DEVOPS

Azure DevOps provides developer services for allowing teams to plan work, collaborate on code development, and build and deploy applications. Azure DevOps supports a collaborative culture and set of processes that bring together developers, project managers, and contributors to develop software. It allows organizations to create and improve products at a faster pace than they can with traditional software development approaches.



REPOS:

The repo was first transferred from the IIHT Gitlab once the project '**PeregrineApp**' was created under the organization '**bcwallet220125z3vlemu3hf**'. From then, different changes were done on different branches and merged to the main branch '**feature/release_v1**'.

Repo Link:

https://bcwallet220125z3vlemu3hf@dev.azure.com/bcwallet220125z3vlemu3hf/PeregrineApp/_git/PeregrineApp

Profile Access Token: w5h5a3qh5ik6bw3sx2xcfxuizzqe5bsefudg5kmkfe4l3thzeyva

Branch: 'feature/release_v1'.

PIPELINES:

There were two pipeline which were created:

- **PeregrineApp-buildAndPush-BE:** creates a docker image for backend and pushes it to the Azure Container Registry
- **PeregrineApp-buildAndPush-FE:** creates a docker image for frontend and pushes it to the Azure Container Registry

NOTE: A service principle was created to make a link between Azure DevOps and Azure Container Registry.

The Docker Compose for back end:

```
version: '3'
services:
  peregrine-app:
    image: peregrinecontainers.azurecr.io/peregrine-app
    build:
      context: ./
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    volumes:
      - /data/springboot-docker-compose-app
```

The Docker Compose for front end:

```
version: '3'
services:
  peregrine-app-fe:
    image: peregrinecontainers.azurecr.io/peregrine-app-fe
    build:
      context: ./
      dockerfile: ./peregrineFE/Dockerfile
    ports:
      - "8080:4200"
    volumes:
      - /data/springboot-docker-compose-app-fe
```

EXAMPLE OF AZURE YML FILE FOR PIPELINE:

```
# Docker
# Build a Docker image
# https://docs.microsoft.com/azure/devops/pipelines/languages/docker

trigger:
- none

resources:
- repo: self

variables:
  tag: '$(Build.BuildId)'

stages:
- stage: Build
  displayName: Build image
  jobs:
  - job: Build
    displayName: Build
    pool:
      name: Default
      demands:
      - agent.name -equals DESKTOP-ROG
    steps:
    - task: DockerCompose@0
      inputs:
        containerregistrytype: 'Container Registry'
        dockerRegistryEndpoint: 'azure others registrycontainers'
        dockerComposeFile: '**/docker-compose-be.yaml'
        action: 'Run a Docker Compose command'
        dockerComposeCommand: 'up --build --no-start'
    - task: DockerCompose@0
      inputs:
        containerregistrytype: 'Container Registry'
        dockerRegistryEndpoint: 'azure others registrycontainers'
        dockerComposeFile: '**/docker-compose-be.yaml'
        action: 'Push services'
```

AZURE CONTAINER REGISTRY

The Azure container registry is Microsoft's own hosting platform for Docker images. It is a private registry where you can store and manage private docker container images and other related artifacts. These images can then be pulled and run locally or used for container-based deployments to hosting platforms.

Login Server: peregrinecontainers.azurecr.io

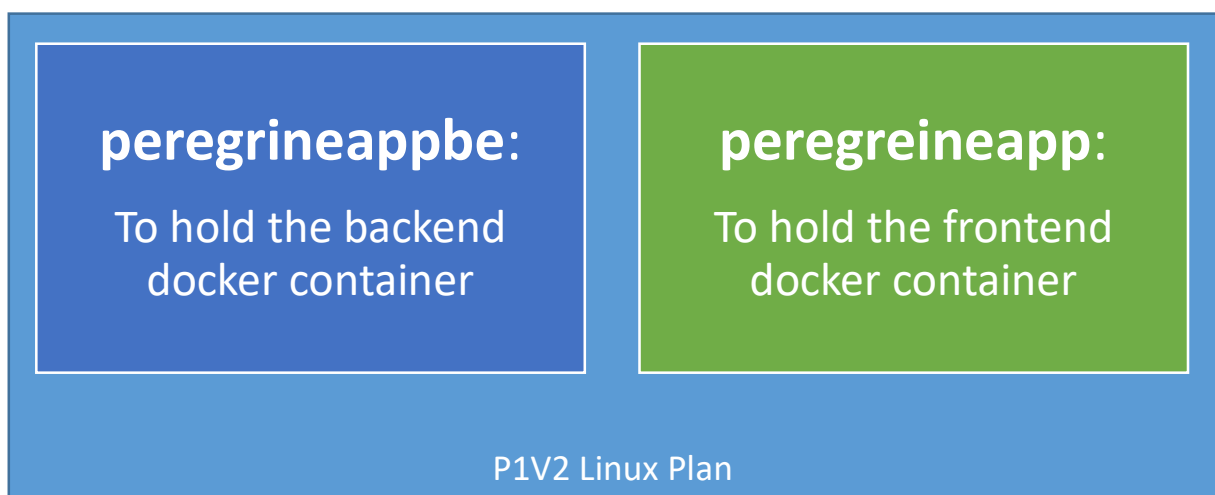
Name: peregrineContainers

Repositories: peregrine-app and peregrine-app-fe

AZURE APP SERVICE

App Service is a managed hosting service for web apps and mobile back-ends. Quickly build, deploy, and scale your web apps either as code or containers. Meet rigorous, enterprise-grade performance, security, and compliance requirements by using the fully managed platform for your operational and monitoring tasks.

Two App services were created under the same service plan.



AZURE EVENTS HUB NAMESPACE

Azure Event Hubs is a big data streaming platform and event ingestion service. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching/storage adapters.

Name of the event hub: peregrinekafka

Application.yml configuration

Following are the changes done to application.yml to link the application to the azure cloud event hub.

```
spring:
  profiles: prod
  kafka:
    bootstrap-servers: peregrinekafka.servicebus.windows.net:9093
    properties:
      sasl.jaas.config:
org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString"
password="Endpoint=sb://peregrinekafka.servicebus.windows.net/;SharedAccessKey
Name=RootManageSharedAccessKey;SharedAccessKey=KYqJc2gTsz5Ayg3ZMAJBul2R/6zMpa8
r9aAJnkDOYeQ=";
      sasl.mechanism: PLAIN
      security.protocol: SASL_SSL
    client-id: peregrineapp
    template:
      default-topic: tweet-events
    producer:
      bootstrap-servers: peregrinekafka.servicebus.windows.net:9093
      key-serializer: org.apache.kafka.common.serialization.IntegerSerializer
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
    consumer:
      bootstrap-servers: peregrinekafka.servicebus.windows.net:9093
      key-deserializer:
org.apache.kafka.common.serialization.IntegerDeserializer
      value-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
      auto-offset-reset: earliest
      group-id: tweet-events-group
```

CHALLENGES FACED

There were many challenges faced with DevOps. I am unable to a service connection in azure devops while linking to the azure web app. Error:

Failed to set Azure permission 'RoleAssignmentId: f60b056b-6fb5-4584-b8e0-1077fd7e2e45' for the service principal '933af9aa-2259-4c44-9bce-45613200a739' on subscription ID '36d70597-f566-43bd-a174-63c25a7ee2c0': error code: Forbidden, inner error code: AuthorizationFailed, inner error message The client 'bc_wallet_220125_z3vlemu3hf@BuilderCloudProvisioning.onmicrosoft.com' with object id '795a9980-e705-4edc-a1a5-8388b7bec8c7' does not have authorization to perform action 'Microsoft.Authorization/roleAssignments/write' over scope '/subscriptions/36d70597-f566-43bd-a174-63c25a7ee2c0/resourcegroups/VisualStudioOnline-C93F2775E91542A2BF814241C1817748/providers/Microsoft.Authorization/roleAssignments/f60b056b-6fb5-4584-b8e0-1077fd7e2e45' or the scope is invalid. If access was recently granted, please refresh your credentials. Ensure that the user has 'Owner' or 'User Access Administrator' permissions on the Subscription.

I had written to the IIHT team but the issue was not resolved. Hence, I just created a docker container registry pipeline for each backend and frontend and manually start the app in azure portal.

CONCLUSION

This project helped me to advance in most of the technologies. Integrating jwt and deploying whole project application and its dependencies into the azure cloud was challenging. The knowledge I gathered after doing this project made me go on to another higher level of understanding of the different technologies. I would like to thank Cognizant Academy and Team IIHT for this wonderful opportunity.

BIBLIOGRAPHY

Have received immense help from:

- <https://www.w3schools.com/js/default.asp>
- <https://www.w3schools.com/html/default.asp>
- <https://materializecss.com/about.html>
- <https://www.w3schools.com/jquery/default.asp>
- <https://www.tutorialspoint.com>
- <https://stackoverflow.com>
- <https://cognizant.udemy.com>