Connect Developer's Notebook

version.1.0

1. Introduction

Welcome to Connect! We are thrilled to introduce a revolutionary platform poised to reshape the social media landscape. Connect offers an all-in-one virtual ecosystem empowering users to control and manage their data across various social platforms within a centralized application. Our vision at Connect is to challenge the dominance of large tech companies over personal data. Our dedicated team aims to bridge this gap, providing users with unprecedented control over their digital footprint.

At Connect, user data and privacy are our foremost priorities. We prioritize providing not just a superior digital experience, but also ensuring the utmost security and privacy with every feature we implement. We believe that users should be able to enjoy, express, and explore information freely, without fear of their personal data being sold, breached, or misused.

By categorizing the digital experience into three main branches—personal, professional, and educational—Connect empowers users to control their environment and content independently. Users can tailor their experience, censoring information, content, and connections as they see fit across different spheres of their lives.

In essence, Connect is a groundbreaking social media platform designed to offer an easy, secure, reliable, and trustworthy digital experience like no other.

Table of Contents

2. Getting Started

Author: Zoheb Hasan, Kamrul Hassan

2.1 Dependencies installations and initializations:

File Directory System:

```
CONNECT
│
├── client
│    ├── build
│    ├── node_modules
│    ├── public
│    ├── src
│    ├── types
│    ├── .gitignore
│    ├── package-lock.json
│    ├── package.json
│    ├── tsconfig.json
│    └── Documentations
│
├── server
│    ├── dist
│    ├── node_modules
│    ├── src
│    ├── types
│    ├── .gitignore
│    ├── package-lock.json
│    ├── package.json
│    └── tsconfig.json
│
├── UML
├── .env
└── connect_documentation.md
```

2.1.a `client`

Redirect to client from the root directory:

```
sample-developer@sample-device Connect % cd client
```

Now that you have directed to client, just simply install all of the dependencies by installing the `node modules`. **Note** that `package.json` in to root directory of `client` already contains all of the dependencies and do not require you add any new dependencies. Thus, you can install all of the dependencies by typing

```
sample-developer@sample-device client % npm install
```

Now that you you have installed all of the dependencies for the client, you can start the react server by just simply typing:

```
sample-developer@sample-device client % npm start
```

After that, you should be able to see a success message like the following:

```
Compiled successfully!
You can now view connect in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.31:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Files successfully emitted, waiting for typecheck results...
Issues checking in progress...
No issues found.
```

Here, note that, we will **not** be running `npm run` build or modify the build directory at all. Build file is meant to be uploaded to the web server for the final deployment.

2.1.b `server`

For the Server, most of the steps are similar except for some differences.

Redirect to client from the root directory:

```
sample-developer@sample-device Connect % cd server
```

Now that you have directed to `server`, just simply install all of the dependencies by installing the `node modules`. **Note** that `package.json` in to root directory of `server` already contains all of the dependencies and do not require you add any new dependencies. Thus, you can install all of the dependencies by typing

```
sample-developer@sample-device server % npm install
```

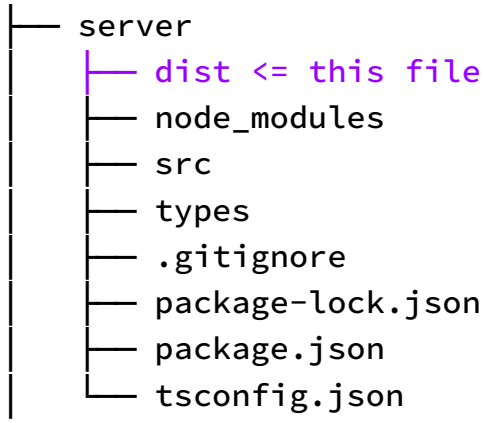
After you are done installing the dependencies, now you need to go to `src` by just directing to the that directory:

```
sample-developer@sample-device server % cd src
```

Now that you are in the you are in the src directory, just simply run the command below to build the executables for the latest build. And you can do that by doing,

```
sample-developer@sample-device src % npx tsc
```

If you notice the directory, you will see that a `.js` version of exectables have been created inside dist directory. Think it as a `a.out` file from `GCC` in `C` language.



Now, all you have to do is to go back to the server directory and just simply run the server by doing the following:

```
sample-developer@sample-device src % cd ..
sample-developer@sample-device server % node dist/server.js
```

And that's all! That should run the server all `port:8000` and you should get this message below in your terminal:

```
The application is listening on port http://localhost:8000
(node:24769) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated
option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and
will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
Connected to MongoDB
```

2.1.c database

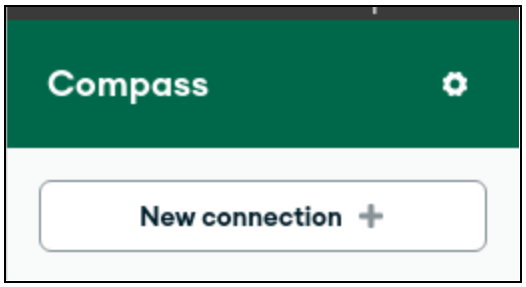
Our database choice for Connect is MongoDB, a non-relational database system. Unlike relational databases that use tables with rows and columns, MongoDB utilizes collections to store aggregate data in the form of documents.

MongoDB Compass installation:

a. **Go to the following link**: https://www.mongodb.com/try/download/compass This link will allow you to download the executable file for MongoDB compass which is just a GUI to interact with our database. By Default it will recognize your operating system.

b. Connection string:

Now that you have installed MongoDB Compass Click on **New connection**



After that, just paste the string below to establish the connection.

```
mongodb+srv://kamrulhassan:fNXADjxipNKubPlP@connect.ny9wvom.mongodb.net/
```

Please do not ever share this string with anyone regardless of the situation.

3. Github (Version Control)

Author: Kamrul Hassan, Zoheb Hasan

Git commands:

You can easily add, commit and push your code using the following commands. Make sure you are always at your root directory so that you get to add all of your updates such as:

```
CONNECT <= this is where you need to be
│
├── client
├── server
├── UML
├── .env
└── connect_documentation.md
```

Now that you are in the root directory of your file, just sequentially run the following commands:

```
sample-developer@sample-device Connect % git add .
sample-developer@sample-device Connect % git commit -m <your message>
sample-developer@sample-device Connect % git push origin <your_branch>
```

It’s always a good idea to check your branch before committing so that you can safely push to your respective branch. You can check that by doing the following,

```
sample-developer@sample-device Connect % git branch
   main
   your branch * <= this  “*” means you are in your branch
```

Pushing code:

Please always push to your **respective branch** first. Afterwards, depending on the project assignment collaborate with you teammate on **Pull Request**. Merge conflicts are bound to happen however, most of the time working on the same task will require you to divide the code into different parts that may or may not conflict. Therefore, communication is key.

Pull Request:

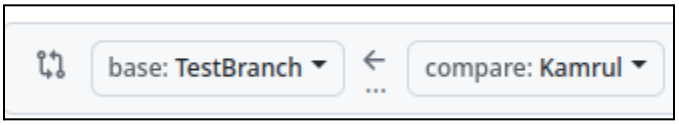
In any scenario make sure to first inform yourself of the recent changes made to the branch. Any changes that occur after a pull request can be overwritten. However, it is best to understand first why the changes occurred. After, you have **written your code** and pushed to your respective branch make sure to do the following:

a. Go to:
   https://github.com/ZohebHasan/Connect

b. Click on Pull requests

c. Click on New pull request

d. Choose the main branch you want to merge from your personal branch



   Note The format is **<main_branch> ← <origin_branch>**

Main Branches:

The **<main_branch>** is usually the following:

a. Server
b. Database
c. Testbranch
d. Conna
e. Security
f. client

4. Security

Authors: Zoheb Hasan

4.1 Connect Web

4.1.a Signal Protocol (End-to-end Encryption)

Connect will be using `libsignal-client` for its End-to-End encryption at scale. The Signal Protocol has been one of the best technologies for modern end-to-end encryption algorithms.

We have already installed its dependencies in our client. If you take a look at `package.json`, you will see the following under dependencies:

File: package.json

```
"@signalapp/libsignal-client": "^0.46.0"
```

4.1.1 Overview

There will be two types of encryption: Message Encryption and Profile Encryption. Both will share the same nature with **some slight differences**.

Messaging Encryption

Key Generation and Management:

Identity Keys: Each user generates a unique identity key pair upon registration.

Pre-keys: A set of pre-keys (one-time use keys) are generated and stored on the server for initiating secure sessions.

Session Setup:

Initial Handshake: When two users start communicating, they exchange pre-keys to establish a secure session.

Double Ratchet Algorithm: Ensures that each message is encrypted with a unique key, providing forward secrecy.

Message Encryption and Decryption:

Encrypting Messages: Messages are encrypted on the sender's device using a combination of symmetric and asymmetric encryption.

Decrypting Messages: Messages are decrypted on the recipient's device, ensuring that only the intended recipient can read the message.

Profile Encryption

Profile Content Protection:

User1, as the root, can send a sender key encrypted with the public keys of User2 and User3.

User2 and User3 can decrypt the sender key with their private keys to access User1's profile content.

User2 and User3 will not be able to see each other's contents unless they follow each other.

Content Encryption:

User1's profile, including photos, videos, and text blogs, stays end-to-end encrypted.

The profile content is encrypted with a unique **symmetric** key (AES).

The symmetric key is then encrypted with the recipient's public key (ECC) and sent along with the encrypted content.

4.1.2 Detailed Steps

Below is a idea on how the encryption, decryption and generation of session keys will take place.

4.1.2.a Key Generation (typescript)

```
import { generateIdentityKeyPair, generatePreKeyBundle } from
'@signalapp/libsignal-client';

      const generateKeys = async () => {

      const identityKeyPair = await generateIdentityKeyPair();

      const preKeyBundle = await generatePreKeyBundle(identityKeyPair);

      return { identityKeyPair, preKeyBundle };

};
```

4.1.2.b session setup (typescript)

```
import { SignalProtocolAddress, SessionBuilder, PreKeyBundle } from
'@signalapp/libsignal-client';

      const setupSession = async (store, address, preKeyBundle: PreKeyBundle) => {

      const sessionBuilder = new SessionBuilder(store, address);

      await sessionBuilder.processPreKey(preKeyBundle);

};
```

4.1.2.c Message Encryption & Decryption (typescript)

```
import { SessionCipher } from '@signalapp/libsignal-client';

const encryptMessage = async (store, address, message) => {
```

```
        const sessionCipher = new SessionCipher(store, address);

        const ciphertext = await sessionCipher.encrypt(message);

        return ciphertext;

};

const decryptMessage = async (store, address, ciphertext) => {

 const sessionCipher = new SessionCipher(store, address);

 const plaintext = await sessionCipher.decryptPreKeyWhisperMessage(ciphertext.body,
'binary');

   return plaintext;

};
```

4.1.2.d Profile Content Encryption (typescript)

Generating Sender Key (Symmetric) :

At first, we will require to generate a symmetric key (AES) in order to encrypt the profile contents.

And Afterwards, as expected, we will be able to encrypt the profile content using that sender key.

Encrypting the Sender Key:

Now that you have the sender key, the next step is to encrypt the sender key using the public keys of the followers using the established session keys

Encrypting the Profile Content: (typescript)

```
import { randomBytes, createCipheriv } from 'crypto';

// Generate a symmetric key for AES encryption
```

```
const generateSenderKey = () => {

   return randomBytes(32); // 256-bit AES key

};

const encryptContent = (content: Buffer, senderKey: Buffer) => {

   const iv = randomBytes(16); // AES requires a random initialization vector (IV)

   const cipher = createCipheriv('aes-256-cbc', senderKey, iv);

   const encryptedContent = Buffer.concat([cipher.update(content), cipher.final()]);

   return { encryptedContent, iv };

};



const videoContent = Buffer.from('Video content of Zoheb playing soccer'); // Example
content

const senderKey = generateSenderKey();

const { encryptedContent, iv } = encryptContent(videoContent, senderKey);
```

Encrypting the Sender key using session keys: (typescript)

```
import { SessionCipher } from '@signalapp/libsignal-client';

// Encrypt the sender key with the session key

const encryptSenderKey = async (store, address, senderKey) => {

   const sessionCipher = new SessionCipher(store, address);

   const encryptedSenderKey = await sessionCipher.encrypt(senderKey);
```

```
    return encryptedSenderKey;

};

// Example usage for Kamrul and Nikhil

const encryptedSenderKeyForKamrul = await encryptSenderKey(zohebStore, kamrulAddress,
senderKey);

const encryptedSenderKeyForNikhil = await encryptSenderKey(zohebStore, nikhilAddress,
senderKey);



// Store encrypted content and keys on the server

await uploadEncryptedContentToServer({

 encryptedContent,

 iv,

 encryptedKeys: {

 kamrul: encryptedSenderKeyForKamrul,

 nikhil: encryptedSenderKeyForNikhil,

 },

});
```

4.2.b TLS Protocol(Transport Layer Security):

Christopher Rosales Kamrul Hassan Zoheb Hasan Nikhil Sundaresan

5. Content Analysis & Moderation

Sean Erfan Nikhil Sundaresan

Content analysis:

- A multi-label classification problem: 1 item can have multiple outputs (e.g. 50 tags instead of 1)
    - Alternative: Create a dense embedding for each image. Essentially converts an image to a large list of numbers. Will allow us to group similar images more effectively, since similar images will have similar embeddings, and it will probably make it easier for us to handle new trends (instead of manually adding new tags and retraining)
- Needs to work on both mobile and web versions (I doubt this matters - we'll probably be using the same model for both)
- Posts can be just text, multiple images+text, or video+text
    - Need to consider how the latter 2 will work. Will images be treated separately or as one unit? And will the text work together with the image?
- It is probably not worth it to start from scratch - Fine Tuning a pre-trained model will be a lot faster and probably more effective
    - This is true for all 3 types of media
- We also need to find a dataset suitable for our task
- TensorFlow lite: Train TensorFlow models like normal, then convert them to make them more effective on mobile devices
- Pytorch mobile: Pytorch's version
- MediaPipe: Helps with certain tasks and provides pre-trained models. Doesn't seem to have video classification but does have text and image classification/embeddings

Boiler-plate code for creating a multi-label image classification model in TensorFlow

```
# Define your CNN model
def create_model(input_shape, num_classes):
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
```

```
        tf.keras.layers.Dense(num_classes, activation='sigmoid')  # Sigmoid for multi-label classification
    ])
    return model

# Example parameters
input_shape = (128, 128, 3)  # Input image shape (height, width, channels)
num_classes = 10  # Number of output classes

# Create the model
model = create_model(input_shape, num_classes)

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',  # Binary cross-entropy for multi-label classification
              metrics=['accuracy'])

# Train the model (example using dummy data, replace with your dataset)
# Assuming `X_train` is your training images and `y_train` is your training labels
# Adjust batch size, epochs, and validation split as needed
history = model.fit(X_train, y_train, batch_size=32, epochs=10, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')

# Make predictions
# Assuming `X_new` is your new images for prediction
predictions = model.predict(X_new)
```

Boilerplate explanation

- Creating model
    - Sequential - A sequence of neural network layers
    - Conv2D - Filters that go through the image and look for certain features (such as edges, points, etc)
    - MaxPooling2D - Reduces dimensions of the image while still preserving important information from Conv2D
- binary_crossentropy - Loss function used for binary classification (which also works for multilabel classification since it can be divided into many binary classifications)

Converting TensorFlow model to TensorFlow lite

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
```

```
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

Moderation:

- I suspect that this is a much easier task than analysis, and that we will be able to figure this out after the analysis problem (like we can literally add a tag for whether this content “needs moderation”)
- I don’t think that context between the text and image is very important for text classification
    - Text: https://huggingface.co/michellejieli/NSFW_text_classifier?not-for-all-audiences=true
        - This seems to have a lot of false positives - Many SFW things are being marked as NSFW, even the word “sad”
    - Images/Videos (at least without sound): https://github.com/bhky/opennsfw2
        - Not tested