CSC301 - FINAL PROJECT
SYSTEM DESIGN

UofTalk™

Boundless Connections

Manav Bhojak, **A**jitesh Misra, **Y**aman Abouyouniss, **P**ranshu Patel,
Abhay Kaushik, **S**harven Prasad Dhanasekar, **S**hayan Iman

Table Contents:

CRC Cards

All the classes below except for Frontend are going to be implemented as Python Classes.

Class name: App
Parent class (if any): None
Classname Subclasses (if any): None
Responsibilities: Handles all communication from the application layer to the presentation layer in the 3-tier Architecture
Collaborators: Matcher, CommunicationHandler, MongoDriver, Authenticator, Frontend

Class name: Matcher
Parent class (if any): None
Classname Subclasses (if any): None
Responsibilities: Determine who this user matches with and fetch those matches from the database.
Collaborators: App, MongoDriver

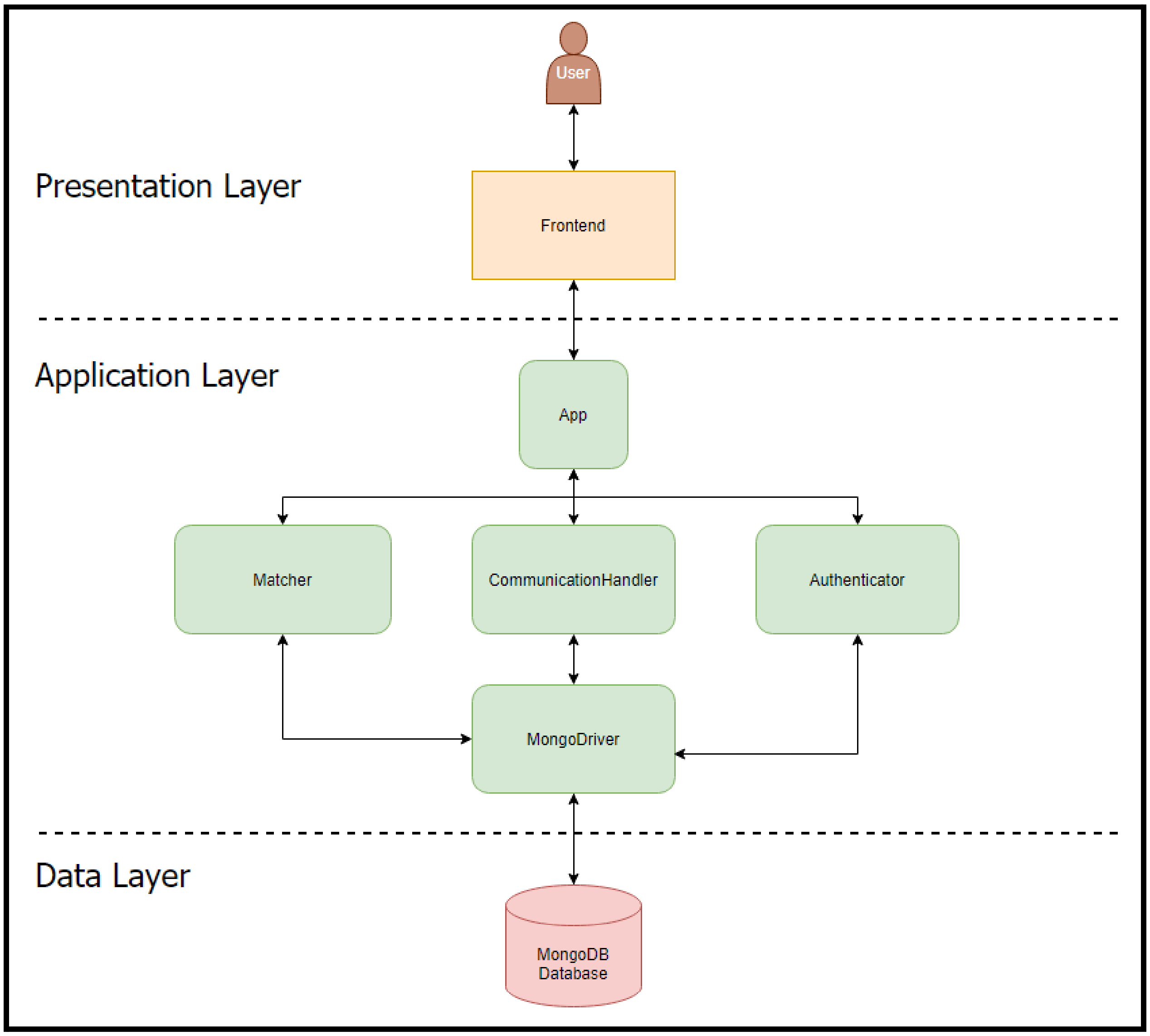
Class name: CommunicationHandler
Parent class (if any): None
Classname Subclasses (if any): None
Responsibilities: Handles message transfer between user clients and network sockets for private/group chatting. Additionally, stores chat logs in the MongoDB database.
Collaborators: App, MongoDriver

Class name: Authenticator
Parent class (if any): None
Classname Subclasses (if any): None
Responsibilities: Authenticates the user's credentials and handles generation of verification codes.
Collaborators: App, MongoDriver

Class name: MongoDriver
Parent class (if any):
Classname Subclasses (if any): None
Responsibilities: Allows application layer classes to interact with the database
Collaborators: App, Matcher, CommunicationHandler, Authenticator

Class name: Frontend
Parent class (if any): None
Classname Subclasses (if any): None
Responsibilities: Serves as the client side view which is loaded by App
Collaborators: App

Software Architecture:



Legend:

Gold:	HTML/CSS/Javascript
Green:	Python Scripts
Red:	MongoDB Database DB

Architecture Overview:

We will use the 3-tier Architecture

1. **Presentation Tier**
    - Contains the front end view of the app
    - Uses Bootstrap, Material UI, React, HTML/CSS/JavaScript, and SocketIO

2. **Application Layer**
    - Provides logic to connect the front-end operations to the backend MongoDB database through Flask
    - Additionally handles operations such as matching users, handling user communications, and sending queries to update/retrieve data from the database
    - User matching uses the following python libraries:
        - NumPy
        - Sklearn
        - Pickle
    - User communications is done using the Flask-SocketIO

3. **Data Layer**
    - Maintains the MongoDB database and the state of the app in physical disk
    - The application will be hosted using Azure Web Service using free UofT student trial
        - Sufficient System Specifications
            - OS: Linux
            - Disk Space: 4GB
            - RAM: 1 GB
            - Server-Client architecture
    - MongoDB database will be stored on disk and will be managed through the python package pymongo

System Decomposition:

Error/Exceptional Cases:

- **Matcher**
  - This component will not receive any exceptional cases because it is purely internal to the system architecture
- **CommunicationHandler**
  - User inputs invalid messages such as empty or non-ascii strings
    - Reject empty/non-ascii messages from users by performing a sanity check on the message
  - If a user's internet connection disconnects, or they close their browser, or they refresh the page
    - Ensure no undefined behaviour occurs
    - Store the entire chat history for a session between users into the database
  - Blocked users exist in the same group chat
    - Messages from the blocked user will not be displayed to the users that blocked them
- **MongoDriver**
  - This component will not receive any exceptional cases because it is purely internal to the system architecture

- **Authenticator**
    - Invalid user id or password
        - Show popup to user indicating this
    - User tries to register using a non-UofT email
        - Reject request and show popup indicating that they need a UofT email to register
    - User tries to register using an existing UofT email
        - Reject request and show popup indicating that the UofT email is already in use
    - User tries to register with a UofT email other than their own
        - Send an email with a unique code to the provided UofT email address and prompt the user to enter the code
- **App**
    - User leaves field empty in the registration survey
        - The user will not be able to proceed with using the app unless they provide an answer
    - User does not complete the meme evaluation
        - The user will not be able to proceed with using the app unless they provide an answer
    - User enters non-ascii input into their bio field
        - Whitelist ascii-input for user bio
    - User deletes their account
        - Remove this user from all their group chats and this user will no longer be displayed as a match to other users
    - User changes preferences while they are matched with someone that they started chatting with
        - The chat session will still be active, but this user will get potentially different matches