

Tinder Clone Design Document

1.0 Introduction

Creating a clone of Tinder and creating with regards to the same usability and functionality as the original Tinder application

1.1. Goals and Objectives

This document describes important aspects of the implementation of how users will be able to match with other users and how user data will be tracked and stored.

1.2. Statement of Scope

Decisions in this document are made based on the following priorities (most important first): Reliability, Usability, Portability, Efficiency

1.3 Software Context

Blocking information will be maintained in the ConnectionToClient objects. The various commands will update and query the data using setValue and

1.4. Major Constraints

Issue 1: Where should we store the information regarding the Users username, password, and user ID?

Option 1.1: We can store the information in a User class, that class will then be uploaded to a User DB, which will then feed into a DataEntryList. This way we will be able to keep track of all user Logins and IDs

Decision: 1 . User information will be stored in a User database (text file), this will include the username, password, and a user ID (which will not be available to normal users). This userid is how the User DB will validate the username and password which is being entered and if it is present with a registered username/password, then the user will be able to log into their account. If not a error message will occur.

Issue 2: How shall matches appear in front of a user in the match screen

Option 2.1: A match queue will be present in which users with the closest location will appear first. We will need to manually enter information about how far each city is from each other. This will then get pulled from the Location DB, and get filtered according to the city which the current user is residing in. And potential matches will be shown accordingly.

Decision 2: Users will be shown a queue of potential matches based upon distance, this distance will be called upon from the Location DB, to see how far a user is from another. The user will then see other users which are closest within their region.

Issue 3: How will we keep track of user choices?

Option 3.1: Once a user has begun to swipe left or right on potential matches, each swipe will be stored in a Match DB. This will keep track of all of the swipes a specific user ID has swiped upon. In order for a match to occur, two user IDs must swipe on each other. When the Match DB is able to verify two user IDs both swiped right on each other, this

will call the match pop up saying Congratulations. Only if two user IDs swipe right then a match will occur.

Decision 3: Every swipe which a user has given will be stored in the Match DB under their specific user ID. Only if two user IDs swipe right on each other, will a match be prompted for both users. If both users swipe left or if only one user swipes right, then this will be stored in the Match DB, under their specific user IDs. Matches will only occur if both user IDs swipe right.

Issue 4: Where will each Profile information get stored?

Option 4.1: Profile information will be stored in a Profile DB. This will be unique to each user ID, and will contain the user's location, age, display name, gender preference, and short description. Once logged out and relogged in, the Profile DB will then access the user ID which logged in and retrieve all of their data.

Decision 4: Profile information will be stored in a Profile DB, this will be called upon when prompted as a potential match. The profile information will include the users age, location, gender preference, picture, and bio. The rater user will only be able to view the ratee's profile information but will not be able to edit it. If a user is prompted to swipe right on the ratee, then the information will be passed to the Match DB.

Issue 5: How will all of the databases interact with each other?

Option 5.1: All databases will be linked to a datastore module, this module will include databases from the User DB, the Profile DB, the Match DB, and the Location DB.

Decision 5: The datastore module will be able to store the User DB, Profile DB, Match DB, and Location DB. This will allow the user to access specific data criteria when prompted by the user filter option.

2.0 Data Design

The classes will be separated into three main categories, which shall be referred to as modules.

2.1 Login Module

- the login module will store user account information in an instance of the DataStore class, named UserDB
- profile information will be stored in another instance of the DataStore class, named ProfileDB
- UserDB will include fields such as: user_id, user_name, user_password
- ProfileDB will include fields such as: profile_id, profile_uid, profile_fname, profile_lname, profile_gender, profile_bdate, profile_picture, profile_pref_agemin, profile_pref_agemax, profile_pref_loorange, profile_pref_gender
- ProfileDB will utilize the user_id to obtain the current user's profile details

2.2 Match Module

- the match module will store match information in an instance of the DataStore class, named MatchDB
- the match module will store distance between locations in the LocationDB for filtering Profiles
- MatchDB will include fields such as: match_id, match_rater, match_ratee, match_option
- LocationDB will include fields such as: location_id, location_initial, location_final, location_distance
- the Filter class will be used to limit results obtained before they are stored in a ProfileList

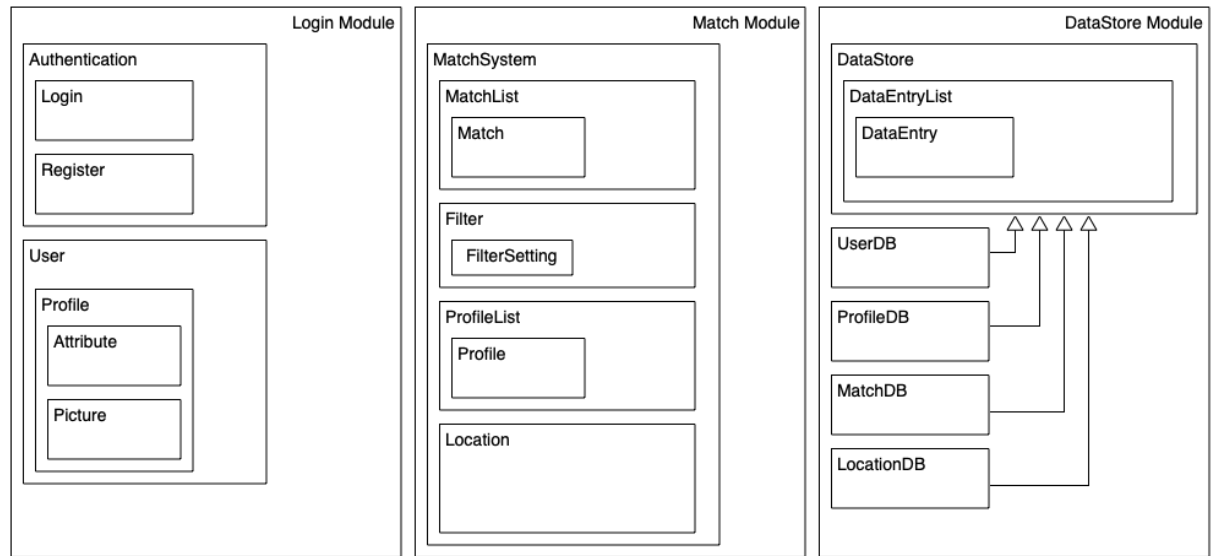
2.3 DataStore Module

- the datastore module will provide the class necessary for data storage, manipulation and retrieval: DataStore
- the datastore module will provide 4 instances of DataStore: UserDB, ProfileDB, MatchDB, and LocationDB
- information will be stored in row format with a series of columns describing each entity
- entries will be retrieved by specifying field value ranges to filter results into a DataEntryList

3.0 Architectural and Component-level Design

3.1 Program Structure

3.1.1 Architecture diagram



3.2 Description of Login Module

3.2.1 Login Module processing narrative

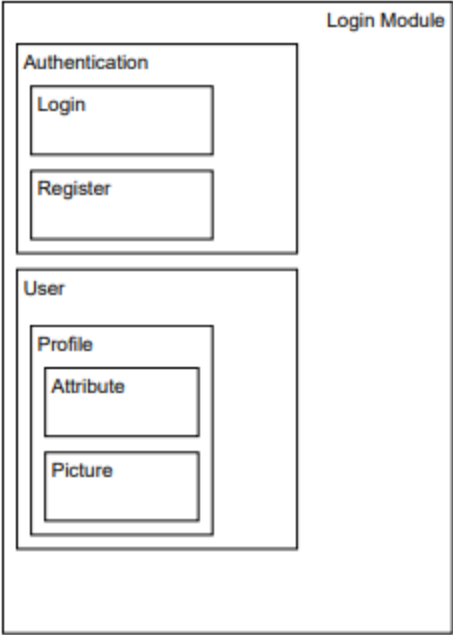
- Provides user login, registration, and profile management functionality.

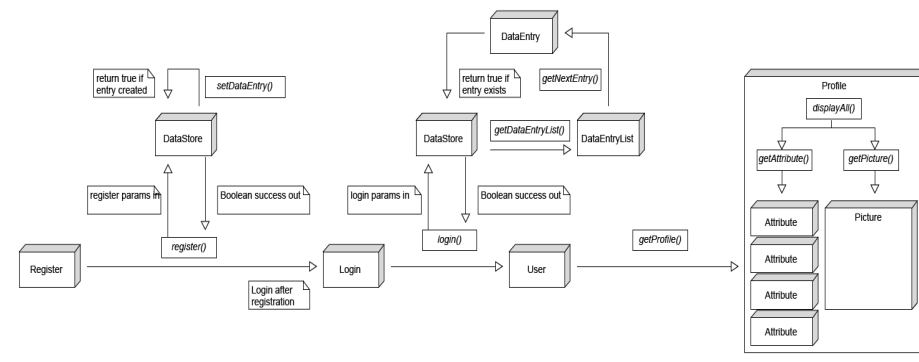
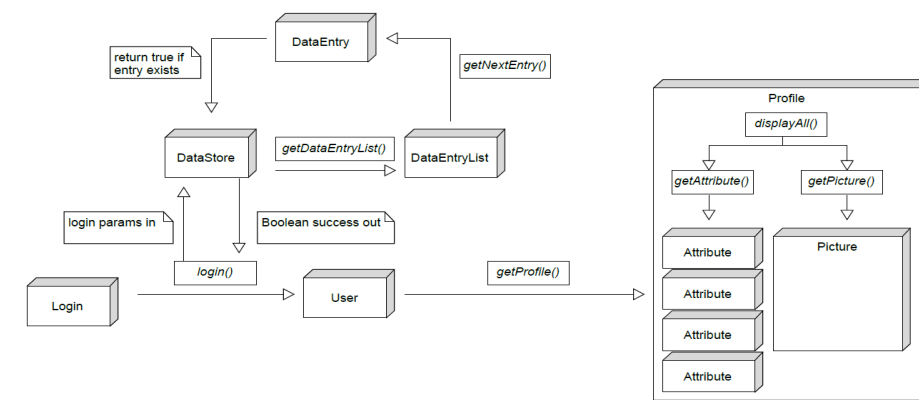
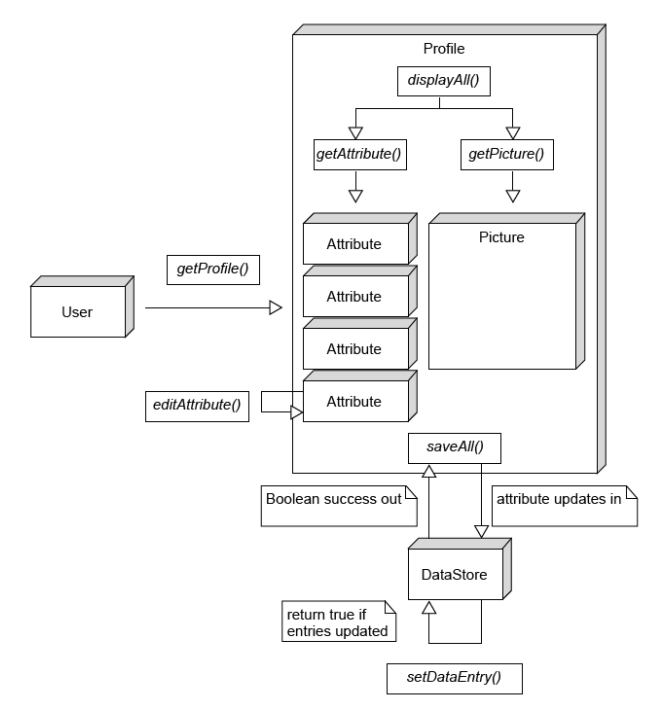
3.2.2 Login Module interface overview

- **login** will take user's credentials
- **logout** will end a user's session
- **register** will take a new user's credentials and save them in user database.
- **isActive** returns if the user is currently logged in

3.2.3 Login Module interface details

Authentication
authenticate(username,password):Boolean setTarget(DBpath):void
Login
isActive():Boolean login(username,password):Boolean logout():void
Register
register(username,password):Boolean
User
getProfile():Profile getAccount():Account getUsername():String getUserID():String isActive():Boolean getRandomLocation():String
Profile
displayAll():void getAttribute():Attribute editAttribute(attribute_name):void getPicture():Picture saveAll():Boolean
Attribute
getUserValue():String getPrefValue():String getName():String setUserValue(value):void setPrefValue(value):void setName(name):void
Picture
getPath():String setPath(path):void





3.3 Description of Match Module

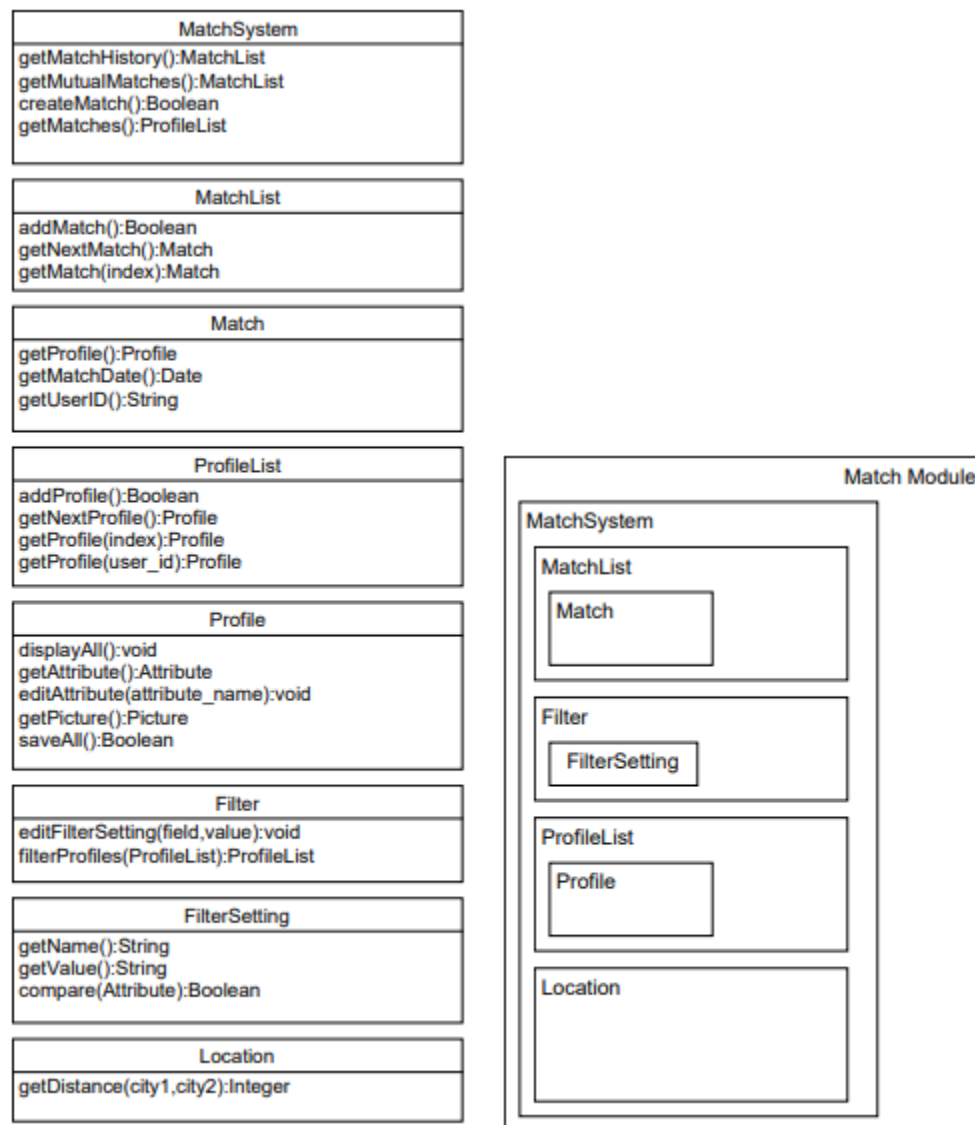
3.3.1 Match Module processing narrative

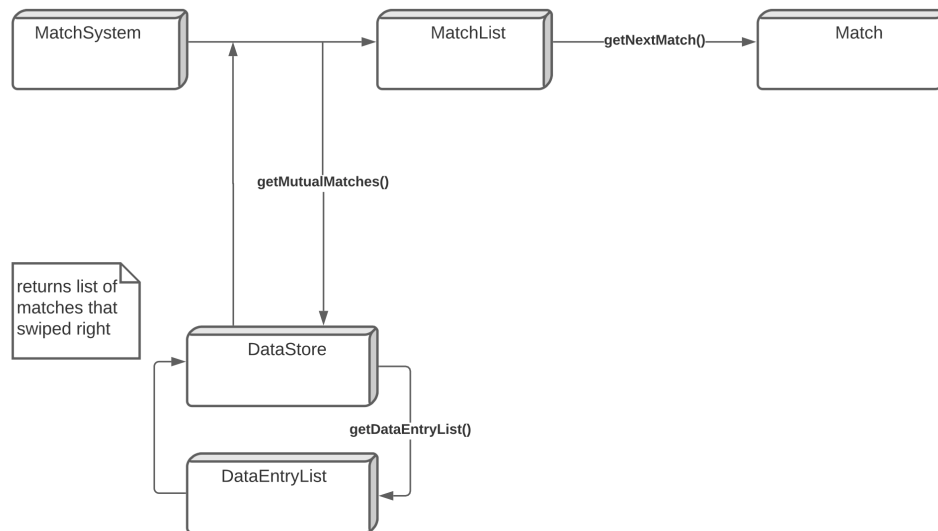
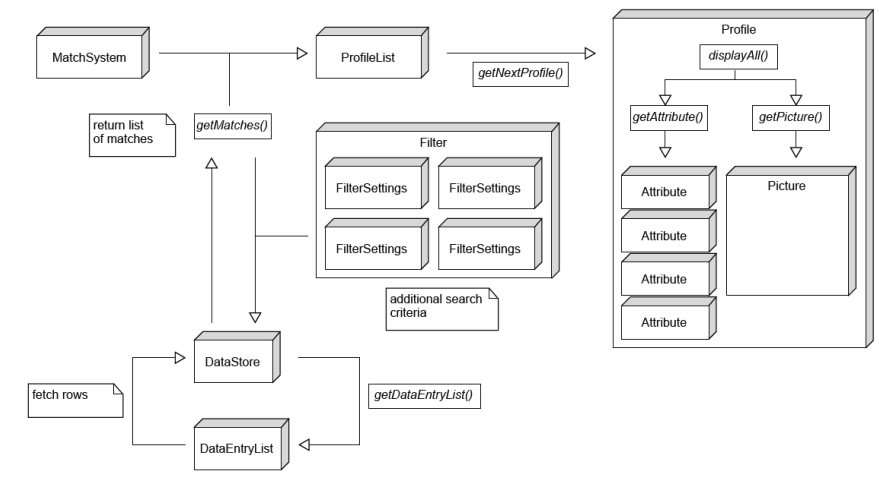
- Match system lets user find potential matches and view matches

3.3.2 Match Module interface overview

- **addMatch** will add a match to the match list that will be displayed to the user
- **addProfile** will add a profile to the profile list of potential matches that are displayed to the user
- **getNextProfile** gets the next available Profile for rating
- **createMatch** adds a line entry to the MatchDB file
- **editFilterSettings** adjusts user preferences used to filter match results
- **getDistance** returns the distance between two cities retrieved from LocationDB

3.3.3 Match Module interface detail





3.4 Description of DataStore Module

3.4.1 DataStore Module processing narrative

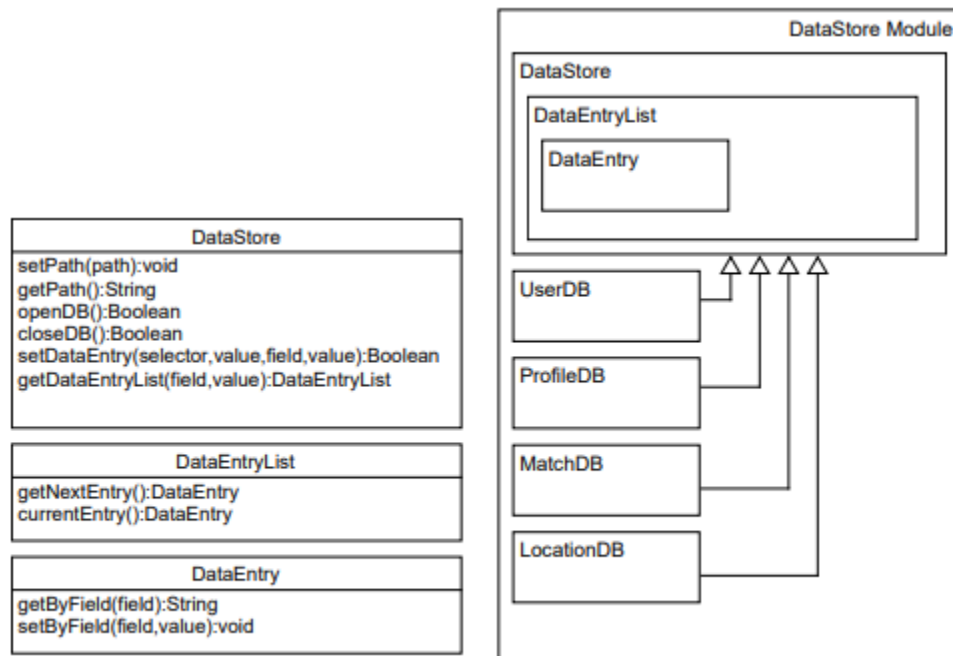
- Accesses and inserts data into databases

3.4.2 DataStore Module Interface overview

- **setDataEntry** will add the corresponding information needed to be stored in the databases

- **setPath** will set path to the corresponding database
- **openDB** will open the connection to the corresponding database
- **closeDB** will close the connection to the corresponding database
- **getDataEntryList** will return the relevant rows of information from the database in the form of a DataEntrylist.

3.4.3 DataStore Module interface detail



3.5 Description of Client and Server Side

3.5.1 Client-Side processing narrative

- Client requests data from the server

3.5.2 Client-Side Interface Overview

- **Datarequest** will allow Client to request information from primary class databases
- **DataValue** will define a value for the data point requested

3.5.3 Server-Side processing narrative

- Server stores requests and returns confirmation to Client

3.5.4 Server-Side Interface Overview

- **DataHost** receives request from Client
- **DataStore** processes database requests

3.5.5 Client-Server Interface Detail

