Logo, company name

Description automatically generated

Project PicMe

Version 1.12

Justin Storms, Andrian Stepanyuk, Wyatt Stohr

Central Washing University

CS380 – Final Project

June 2, 2023

Table of Contents

[Project Description 4](#_Toc136817320)

[User Guide 5](#_Toc136817321)

[UML Diagrams 7](#_Toc136817322)

[Use Case Diagram 9](#_Toc136817323)

[Testing Overview 10](#_Toc136817324)

[Database Testing 10](#_Toc136817325)

[Repositories 10](#_Toc136817326)

[Services 11](#_Toc136817327)

[GUI Testing 12](#_Toc136817328)

[Entity 12](#_Toc136817329)

[Http 12](#_Toc136817330)

[Known Bugs 12](#_Toc136817331)

[Possible Feature Additions 13](#_Toc136817332)

[Login Functionality 13](#_Toc136817333)

[Picture Upload 13](#_Toc136817334)

[Post Upload 14](#_Toc136817335)

[Comment Upload/View 14](#_Toc136817336)

[User Settings and Password Management 14](#_Toc136817337)

[Friend List Management 14](#_Toc136817338)

[Friend Profile Navigation 14](#_Toc136817339)

[Extra Security Measures 15](#_Toc136817340)

[Administrator Views 15](#_Toc136817341)

[Administrator Database Management 15](#_Toc136817342)

[Appendix 16](#_Toc136817343)

Table of Figures

[Figure 1: PicMe Home Screen 5](#_Toc136817344)

[Figure 2: PicMe Photos Screen 5](#_Toc136817345)

[Figure 3: PicMe Profile Screen 6](#_Toc136817346)

[Figure 4: GUI and View Controllers 7](#_Toc136817347)

[Figure 5: Database Queries and Controller 8](#_Toc136817348)

[Figure 6: PicMe Use Case Diagram 9](#_Toc136817349)

[Figure 7: PicMe Database ERD 16](#_Toc136817350)

[Figure 8: PicMe UI Blueprint 17](#_Toc136817351)

[Figure 9: Admin Login UI 18](#_Toc136817352)

[Figure 10: New Admin User Creation UI 18](#_Toc136817353)

[Figure 11: Administrator Main UI 19](#_Toc136817354)

# Project Description

Project PicMe is designed as an application which can be used to network personal information, specifically photos and reactions to those photos. It is meant to connect people through their experiences, as the basis for a social media platform’s functionality. The current version is a prototype of the databasing, and Generic User Interface (GUI) event handling required to accomplish this goal.

The PicMe prototype consists of two distinct halves: the database back-end, and the GUI front-end which the end-user is exposed to. This is connected by controller framework which handles the database queries as well as GUI controllers which manage scene and data preservation between views. Although the prototype does not have the full functionality to connect users it does demonstrate how these profiles can be built to retrieve data from the database to populate a user’s photos and associated comments. This version demonstrates the simplicity of navigation that an end-user can expect from future versions.

Rather than direct SQL statements designed to query a local database, a more adaptable methodology was used because of the non-local nature of profile connections in future development. While the current prototype does communicate with a local database, the framework is in place to communicate with an online database server, allowing for the development of an actual social media platform. SQL requests are made to an API which requires a Json to Gson conversion, and responses convert back from Gson to Json. Because of how the database needs to hold more complex data types, such as .jpg files which are stored as Blob data types, Json returns the Blob instead as a base64 string which requires parsing to write the image to a local directory. String, integer, and date data types are significantly less complicated and are almost directly 1:1 conversion in these requests. However, being able to adapt these different data types during communication allow for non-local database communication.

A third section of the project also exists in development; however, it could not be released for demonstration due to issues with development kits. This is the Administrator interface, designed to allow tickets on database-related manipulation, exporting, and reporting to be processed on the business-end. It also enforces accountability via access and activity logging due to the sensitive nature of the user data stored within the PicMe database.

# User Guide

To install PicMe, simply place the PicMe.zip file in the directory you wish and extract the files. The executable file will be labeled as PicMe. It is recommended not to manipulate the project files otherwise, as the prototype version does not remove them from the user directory and are accessible.

When Home is pressed, PicMe will display a demo Home screen. Know that future versions of PicMe will have more profile controls and security/authentication before arriving at the home screen for a user. On this screen there are a few locations of note:

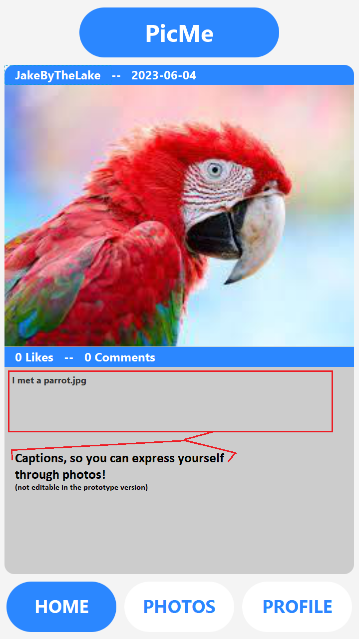


Figure : PicMe Home Screen

Next, we have the Photos screen. All photos from the user are present as thumbnail views. Selecting a thumbnail will update the Home screen with the selected photo, date received, and post caption then navigate the user to the Home screen where it can be viewed.

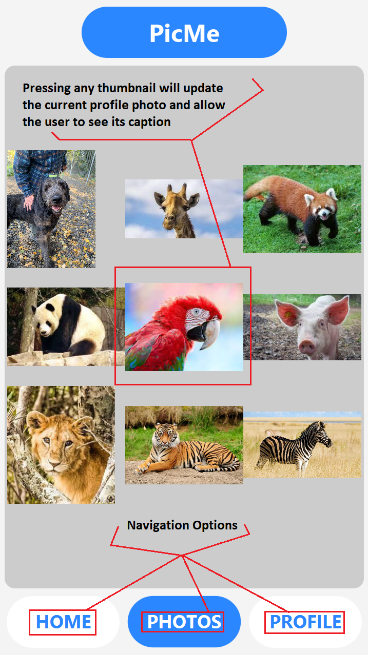


Figure : PicMe Photos Screen

Last is the Profile screen. Here the current picture selected for the Home screen will also be visible, but PicMe will display the user’s profile information instead of the photo caption.

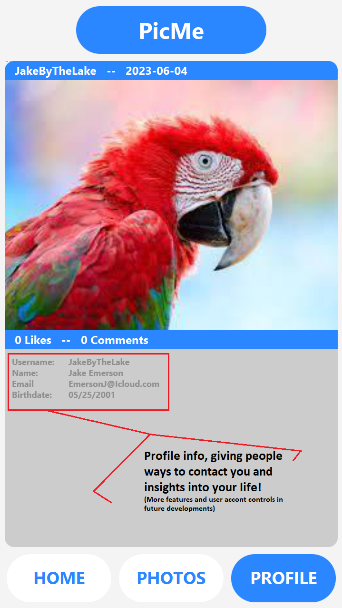


Figure : PicMe Profile Screen

# UML Diagrams

A picture containing text, screenshot, diagram, parallel

Description automatically generated

Figure : GUI and View Controllers

A picture containing text, screenshot, diagram, parallel

Description automatically generated

Figure : Database Queries and Controller

# Use Case Diagram

Note that the use-case reflects friend connection features not yet implemented. However, this is a representation of PicMe use extending to other profiles. It also reflects the request timing for database retrieval.

A picture containing sketch, diagram, drawing, line art

Description automatically generated

Figure : PicMe Use Case Diagram

# Testing Overview

The PicMe application was tested in its two zones: the back-end database, and the front-end GUI. In almost all cases Junit testing was utilized as the test method.

## Database Testing

The database consists of a SQL server which communicates via Http requests. Rather than direct SQL calls from the local database, it is instead designed to move the concept seamlessly to an online database which would process the requests in a similar fashion with minor changes to the methods to call that data. While not all functionality is present in the prototype, the database was tested as new data pull requests were necessary during development. These were designed to be expanded upon as the project continues in development. While not all testing listed are for processes the current prototype utilize, testing was still created for sections still being developed, such as the additional features which will be listed later and the Administrator interface.

Testing was implemented to verify that the various necessary operations for interacting with a database were accounted for. These tests include:

### Repositories

General PicMe Application Testing (database)

This testing checked that Person objects (the basis for a user profile) were able to be called from the database. This provided an additional check which validated how the main primary key in the database (int personID) was being recognized and showing no primary key violations. The console output verifies that the Gson to Json requests provided usable data types and whether further data conversion was needed.

Admin Repository Testing

These tests cover the Admin repository where the usernames and passwords for administrators are stored. Affected areas: SQL insertions of new users, username lookups by ID, and account deletions. The prototype for demonstration did not call for extensive testing since the Administrator interface was a cut extra feature for this deadline.

Comment Repository Testing

Comment repository testing was cut from development with the comment system since other profiles were not able to be navigated in the current build of the project.

Friend Repository Testing

Despite other profiles not being able to be implemented in this prototype, there are methods which update the database entity which governs friend profiles. Addition, deletion, and find functions passed, which are the current scope of operations needed for that entity to develop friend connections in future features.

Person Repository Testing

Person, or the commercial end-user profile base, repositories test for the ability to add, delete, and find the data to make Person objects. ToString validates that the proper data is being stored and that the correct personID reference is made.

Picture Repository Testing

Pictures underwent repository testing to ensure that their unique typing (Blob) as stored in the database was being properly requested. Failures were found when trying to communicate changes as Blob data types. This data type is used in SQL databases to hold image binary data, and when requested through RESTful API to Http, the Json to Gson request does not allow for this method to be read as a byte array (shown as test failure). Because of this, the file itself was transmitted directly through Gson conversion, which was successful.

PostPicture Repository Testing

Posts associated with pictures were tested to ensure the Picture entity’s primary keys could be properly referenced and returned as foreign keys in the PostPicture database relation. These post IDs could return post IDs which can be used with the Post Relation to return post text.

Post Repository Testing

Post repository testing checks for Person IDs to be able to reference the foreign keys in the Post database entity and return any post ID associated with that user. The methodology to find a post by pictureID (as primary key) was tested along with the ability to save a post through this relation. This is an important relation for the current and future development as this key associates pictures to their comments and posts.

### Services

Admin Service Test

Simple test performed to see if the Admin side of the database could have entries deleted from it. Further testing was shelved due to time constraints as the Administrator interface was backlogged.

Friend Service Test

Friend Service testing was focused on the relation between profiles. While the prototype does not make this communication, it is important for future development. Friend-of and friend-to relations are called by personID and allows for the basis of the network graph to be developed in the future. Testing was performed specifically on the ability to reference IDs through this relation, as well as perform options like saving a friend (adding a tuple to the database with friend information) as well as removing that friend relation.

Person Service Test

The ability to update, delete, and find a profile (Person entity) by personID was tested. Aside from the heavily utilized “find” methodology which allow accounts to be presented through the PicMe GUI by retrieving a person object from Http, being able to update these profiles and delete accounts by removing person tuples and all related data relations via these requests were also prepared for future development.

Post Service Test

Tests were performed on the ability to find all posts by personID, as well as the ability to delete posts for future development.

## GUI Testing

Due to the conflicts caused by the modular nature of JavaFX, Junit testing was not able to be performed on the GUI directly. Instead, the Entity, Http, and State packages were checked to ensure that functionality was not lost when transitioning from the Junit test housing in the PicMe1.12 database application to the PicMe GUI directories.

### Entity

The entity classes, which represent objects that hold person, post, and picture data members were tested. The ability to convert Json data from the Http front end into data for output was tested for each data member.

### Http

The controllers for making Http requests were tested, as they are meant to return objects of entity types. Due to the nature of Gson to Json conversions in gathering Http front-end data, this testing was directly tied to the entity testing, as that testing is not possible without a way to create the object to begin with.

# Known Bugs

While the prototype is stable, there are known bugs in the system. Most are related to failed test cases within the database. These include being unable to save a specific person through the current means of data insertion due to a not-null data integrity violation involving the user’s email due to a uniqueness constraint. In a live alpha test, or with more availably created accounts this bug may not present itself in live runs of the database. Saving a post to the Post entity also causes this bug.

There is also a performance bug in the GUI side of the application, where navigation to the traffic-heavy photos page causes undesired slowdown. A known fix for this issue is saving the states of the picture objects which create the thumbnails and provide picture-sensitive information such as captions and dates through the DataSingleton class which allows for data transfer between views via singleton design pattern. Keeping this data in this manner would require checks for file differences between the picture directory and the database but would not require full retrieval and increase application responsiveness when data communication requests are not needed.

There is also a major bug in the IDE where module-based development libraries such as JavaFX cause critical failures with non-module database libraries. As such, Junit testing is not directly compatible with the GUI implementation. Instead, classes which store and interpret data are direct replicas of classes built for data field serialization in the PicMe1.12 database application. Tested, known good retrievals are mirrored into connections set up in the GUI. Effectively, the nature of serializing the data members of object classes built from database calls are identical. Therefore, for completeness’s sake the fields should be tested once IDE conflicts can be resolved.

Additionally, the Administrator Interface and related classes currently have no testing performed on them. This is related to the major bug with IDE modules, as well as a Java Virtual Machine (VM) path error that cannot be currently resolved stopping code from being run at the compiler level. With additional time and manpower this could be resolved, but days of research went into resolving this conflict and it had to be shelved to meet the deadline with a working prototype of the Commercial-user application.

# Possible Feature Additions

Given the result of this early prototype and an underestimation of the manpower and time required to complete the task, there are many future additions which could be added to project PicMe. However, each addition makes the next easier to implement. This is due to the main barrier of feature addition and implementation being translation of the database request to useable output, due to the Gson to Json conversions (and vice-versa) required to create the requests which interact with the database, along with the database data types for data such as images not being directly compatible as objects with GUI outputs such as ImageView objects. The list includes the following (See Figure 2 in appendix for a visual representation):

## Login Functionality

Currently, the demo version of PicMe only supports a single profile navigation, though the database framework exists which supports multiple user account information. Account credentials will eventually be stored upon login, and the user ID functions as the primary key relating directly to many entities in the database. It can also be used retrieve additional primary keys from indirect entities which connect all data associated with that user. The main development needed are the views within the GUI and additional controllers within the entity, http, and state packages within the GUI directory. Most database request code necessary to pull and validate a user already exist.

Additionally, account creation is necessity to this, which will use existing methods used to upload test accounts into the database. Instead of static values, these values will be replaced with references to textField objects the user can easily manipulate within GUI.

## Picture Upload

PicMe actually uses the necessary code to upload data to the database to build the tests and demo profile. As with login functionality, the majority of what is needed to implement this are views and scenes built into the GUI. A local directory where user photos can be stored would be used to determine photos to upload. This way a user will not have to figure out where it is stored, but instead save it to the known location of PicMe in an easily identifiable folder.

Due to the way the Blob data type pulled form the database is required to store pictures; extension validation is also needed. When a picture is pulled, it has to be written as byte data to a file, which will need to be named and appended with an extension to identify file type. The database already holds the necessary data for this, and the controllers are built with data fields needed to hold the String parsing to compare them and write output appropriately.

## Post Upload

As with the prior upload feature, post upload is a simplified version which only requires a string input to be stored as VarChar data in order to implement this feature. A text field capturing the post a user has made regarding a specific picture would run an insert as a new tuple into the Post entity within the database. The view in the GUI would represent the majority of the work needed to accomplish this.

## Comment Upload/View

A comment upload is in many ways similar to a post in terms of the database interaction. It can be associated with either a post or a picture and is retrievable using the person or post ID as a primary key for lookup.

This will allow for polymorphism to handle sorting out where a comment would be assigned within the GUI views for output once pulled. The ID used as a key will determine whether it is a post, or a picture and the same output method may be used in implementation.

## User Settings and Password Management

The user control of their own profile and settings closely resembles actions taken by another future implementation of the PicMe project, the Administrator interface. The view and associated controllers will need to be developed to form the skeleton of this interaction, but it would borrow the already tested database interaction methods for uploading and downloading changes. While obvious, the timing of developing this feature would have to come after the Account Login feature as it would require validation that a user is only able to manage their own data, including the option to change user password.

## Friend List Management

While the networking for the discovery of friends through adjacency via graph theory is complex, the structure of how that data can be stored within a profile is not. Friends are already tracked through a Friend-of-Friend-to entity relation in the database. When a profile is logged in, this list can already be populated creating an immediate friends list. Discoverability through graph calculation will have to be adjusted during beta testing, as too much discovery from “friend nodes” too far away from the initial friend list loses it meaning, while single step navigation though the graph would yield a limited number of results that could prove to be detrimental in nurturing new connections. Additionally, the “distance” between friends on the graph can be weighted by similarities compared upon between user profiles, allowing for nodes physically farther away to be considered as potentially returnable as a possible friend to request.

The user would interface with this idea using a list abstracted form it, populated by a view in the GUI. Selecting a profile and committing to sending a request would allow for verification of connection. The user themselves would be able to see their own request view, where they would approve or deny profiles who send requests.

## Friend Profile Navigation

Friend profile navigation is the resulting functionality of the list of person IDs stored as friends within their profile, retrieved at login. This feature actually would reuse the code for views of the current profile, with notable exclusions of access to account data management and personal data marked private if not verified as the actual user. This would allow the user to navigate a profile as though it were their own, resulting in less time learning the interface.

## Extra Security Measures

While a password itself can be a preventative measure in limiting unwanted access to accounts, both the Commercial and Administrative users will also require a security token input to verify the user. This will be required upon account login and password change. User-generated passwords may not follow good practices to prevent infiltration. A rotating key via token is necessary to reduce the event of brute-forcing login attempts. Up to one billion passwords per second (potential worst case) can be sent by computer as instructions and will try all intermediate values aside from “dictionary entries” commonly found in most passwords. The token’s rotation helps to invalidate brute for attempts up to a certain number of instructions per second. The administrator interface within the PicMe project already enforces a user password with a strength of 72 billion combinations. This is meant to be combined with authentication failure lockouts and prompts to update password with sufficient suspicious logins exist.

## Administrator Views

While mostly complete, the administrator interface still requires testing as the commercial end of the project aligned with the demo on how the prototype could be used as the basis of a social media platform. GUI and controllers are established, but without testing it cannot be released accompanying the rest of the demo. This feature will probably require the least time to bring online but is not a priority until features of the Commercial-user PicMe interfaces are complete. Admin GUI design can be seen below as Figure 3, Figure 4, and figure 5 in report Appendix.

## Administrator Database Management

This feature, while mostly complete being an extension of the Administrator Views, would need to undergo redevelopment if the structure of the Commercial-user interactions changes needs within the database. Therefore, it is better to redesign this in as few iterations as possible once the other aspects of the project are more established.

The administrator database management operates on a different scope than user database manipulation. While the goal of the Commercial-user is connection and sharing, the database admin would use this interface to process ticket requests on users. IDs can be used to pull various user information for export to a secure location. This is in anticipation of legal requirements for the data handling and requests such as law enforcement compliance. All admin user activities are tracked as logs describing the interaction for every execution, upholding accountability and would be output to a secure location for storage. These features already exist within the project structure, but changes may need to be made to implementations as the features on the Commercial end develop.

# Appendix

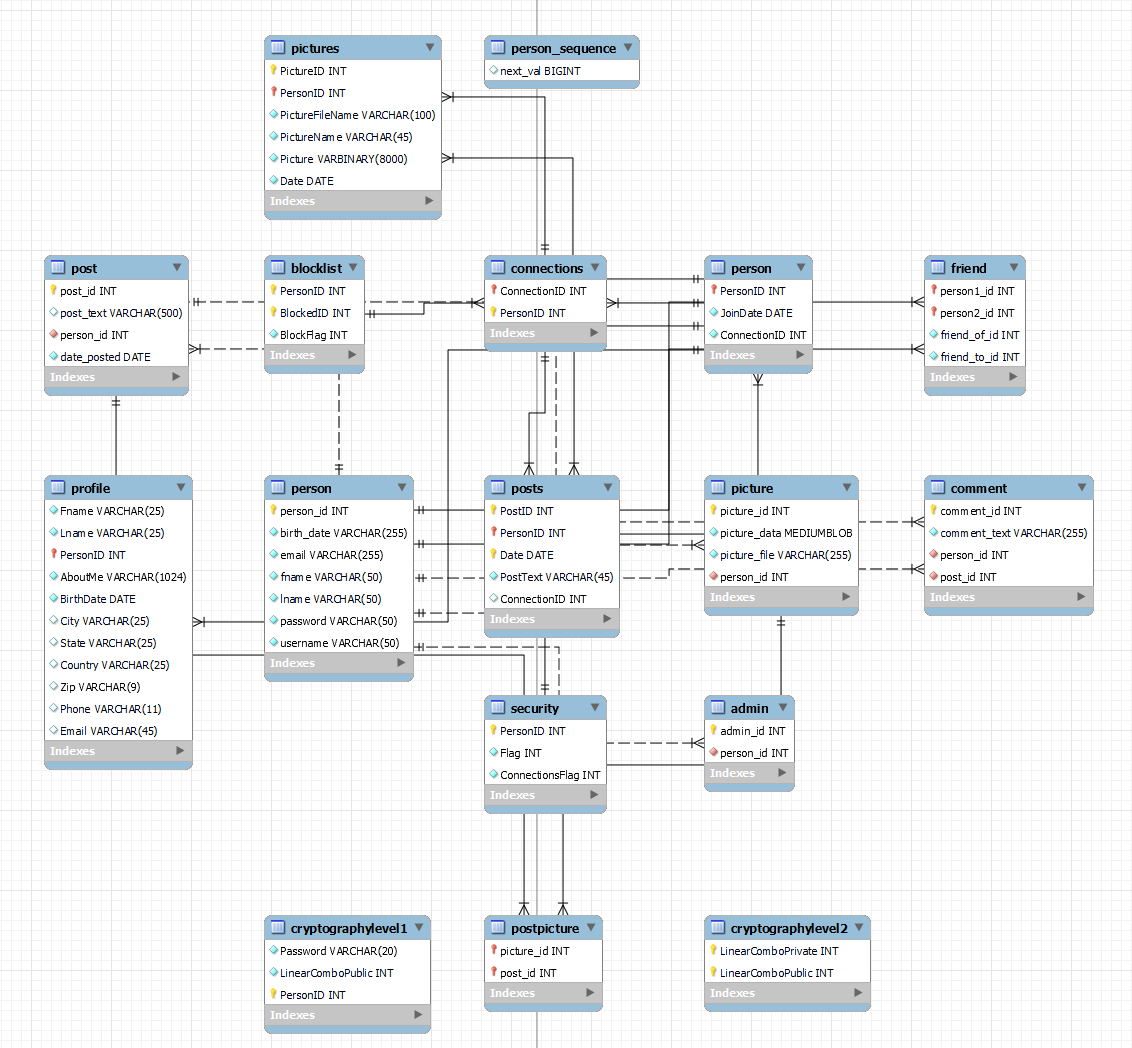


Figure : PicMe Database ERD

A picture containing text, diagram, map, plan

Description automatically generated

Figure : PicMe UI Blueprint

A screenshot of a computer

Description automatically generated

Figure : Admin Login UI

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure : New Admin User Creation UI

A screenshot of a computer

Description automatically generated

Figure : Administrator Main UI