ABFMJ-SecureDove

CptS 428 Software Security Final Presentation and Demo

Members: Muhammad Ali Dzulfiqar, Flavio Alvarez Penate, Phearak Both Bunna, Jaysen Anderson, Mitchell Kolb

Project Background

Background:

SecureDove is an instant messaging app that provides secure messaging with explicit confidentiality and integrity defenses (e.g., checking and defeating message hijacking)

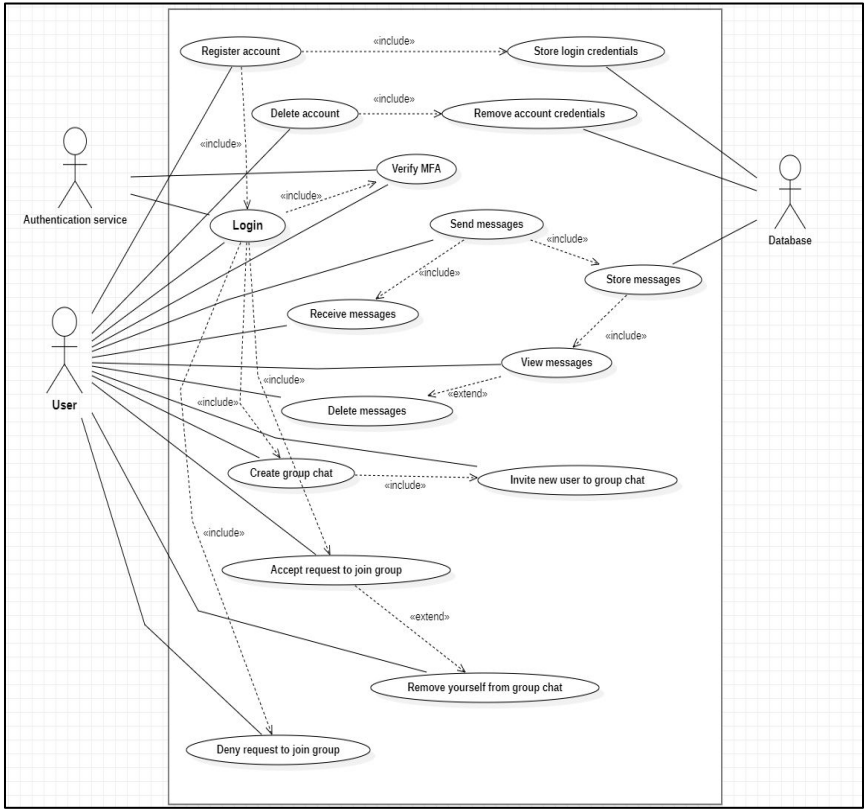
Objectives:

- Provides secure communication between sender and receivers.

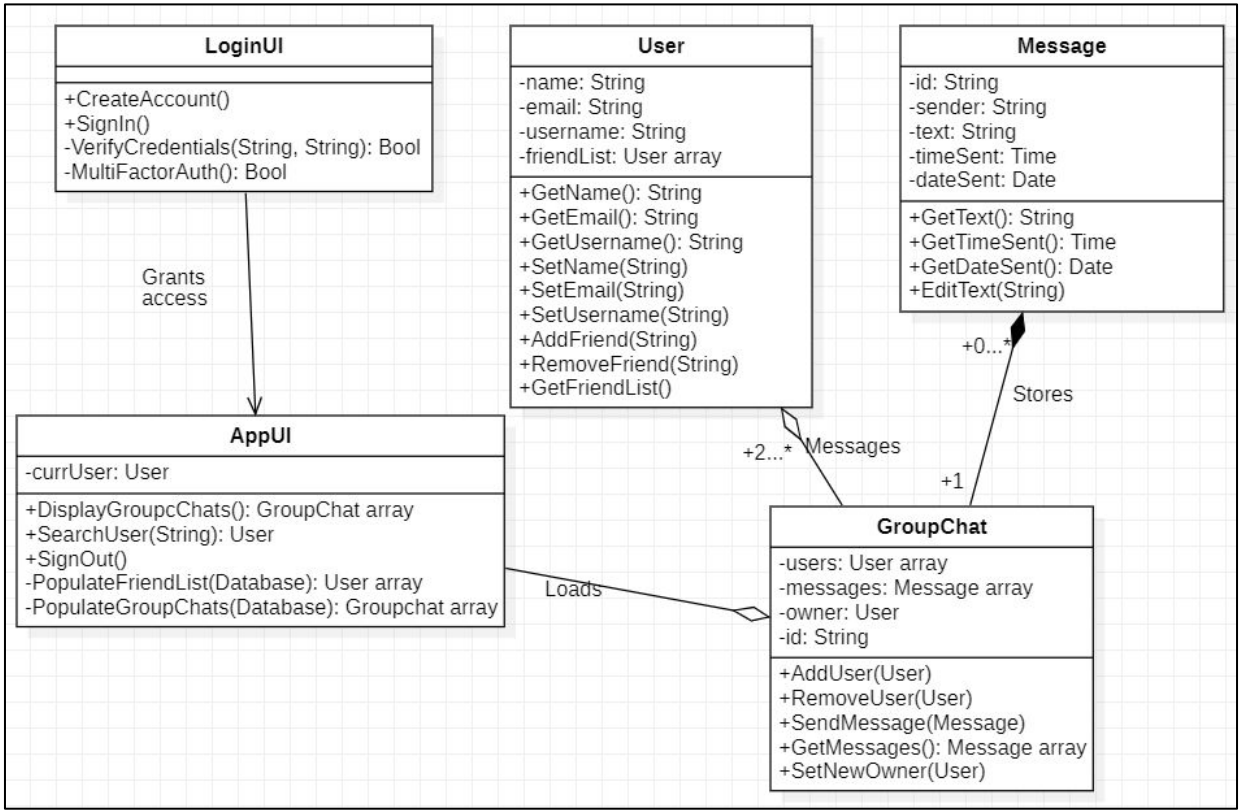
Project Requirements

- User Authentication
- Ability to send and receive message

Use Case Diagram



UML Class Diagram



Security Quality Plans

- Security Goals
- Security Metrics

Security Goals

Confidentiality

- When messages are deleted, they are only deleted for the party that wants to delete them (not all parties)
- Account names are unique to avoid messages going to unwanted users
- End to end encryption
- Multi factor authentication for login
- Users added to a group chat after creation shouldn't have access to previous messages.
- Only the account owner should be able to view messages sent to or from the account

Integrity

- Message text should not be modifiable by a third party or anyone other than the user that sent them.
    - For an edited message it should be marked as "Edited" in the UI for the application.
- Other message aspects, such as time sent, sender, etc, should not be editable by any source (not even the user).
- Users shouldn't be able to be simultaneously logged into multiple accounts.

Availability

- The application should be able to resist denial of service attacks and be always available to the user.
- The application should not provide the user with any frustrations with regards to logging in and making use of its services (i.e. security shouldn't come at the cost of the user's ease of use).

Security Metrics

Confidentiality

- The percentage of how many cases where the messages that got sent are viewed by another person who is not the sender or receiver should be 0%.

Integrity

- The percentage of how many cases where the messages have their body modified due to malicious attack, software or hardware failures after being sent by the sender should be 0%.

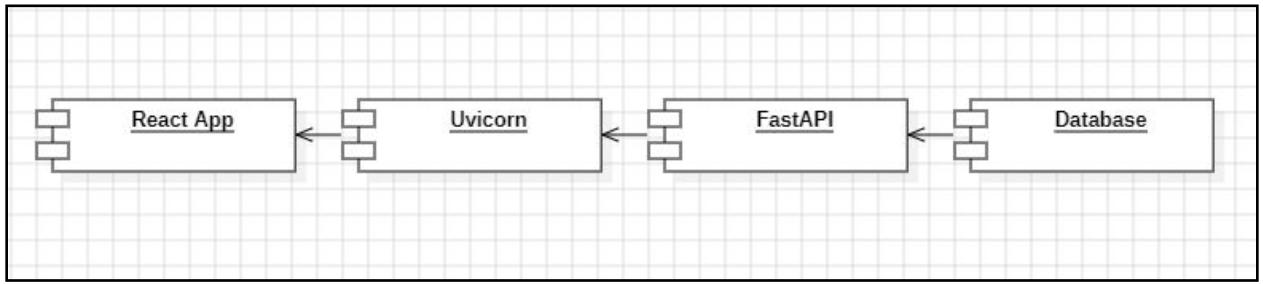
Availability

- The percentage of messages that are lost due to system failures should be 0%
- The percentage of messages that are deleted by a user who is not the sender or receiver should be 0%

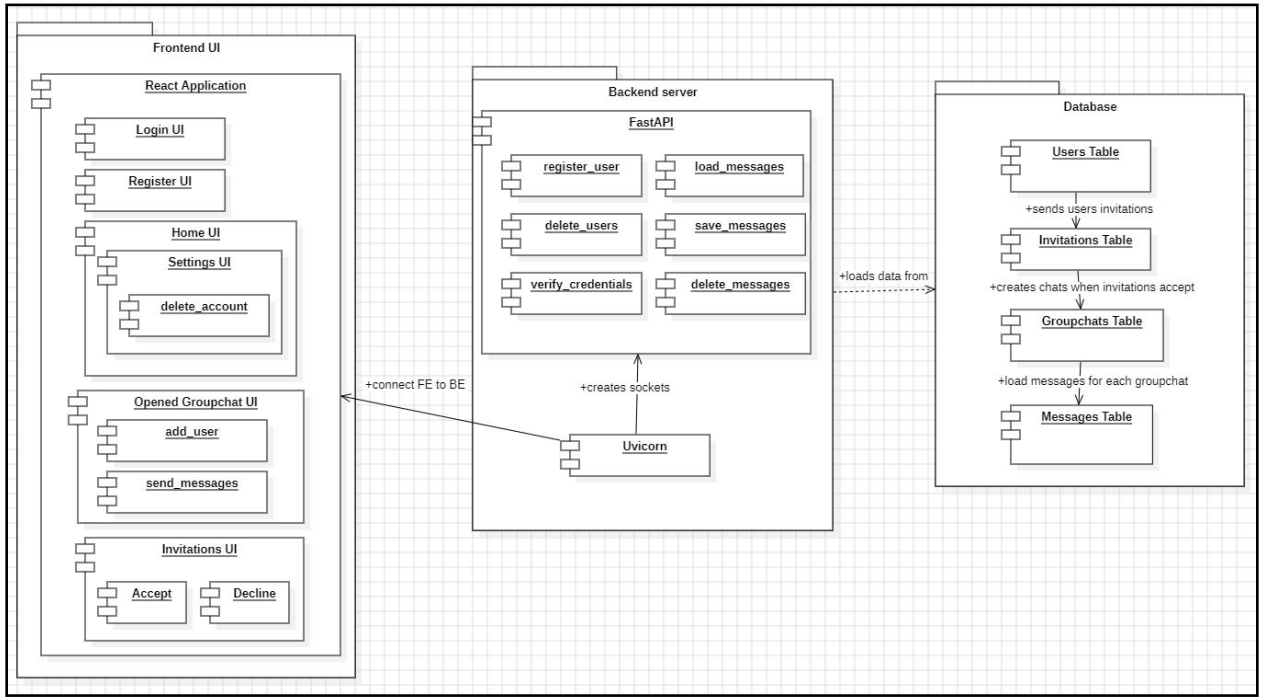
Solution Approach

- Architecture Design
- Component Level Design
- Technologies Used

Architectural Design



Component Level Design



Technologies Used

Frontend: ReactJS

Backend: FastAPI

Version Control: GitHub

Prototyping and Design: Figma

Founded Vulnerabilities

- Database API Key Being Visible
- SQL Injections
- Brute Forceable Passwords
- Passwords are transmitted & stored in plain text
- Known Security Vulnerabilities in dependencies
- URL Open Redirection Vulnerability

Database API Key Being Visible

- The ElephantSQL database key was stored in the backend Python file initially
- This is bad practice since through this key, anyone could edit and tamper with our database
- **Solution**:
  - Store the key in a .env file that is only found inside of our local computers and that is not uploaded to Github.

SQL Injections

- All of our SQL queries were susceptible to SQL injections.
- This could result in catastrophic failure for the program as a simple DROP TABLE command could have wiped out our database.
- **Solution:**
  - Parameterize all SQL queries in the code base.
    - Parameterizing the SQL queries means that all queries will be sanitized rather than directly using the user input in our queries.
    - This ensures that SQL injections will no longer take place.

Brute Forceable Passwords

- There was no password requirements at registration
- A user could enter a password as short as one character and sign up with it
- **Solution**:
  - Require a minimum character limit for passwords during registration
  - Require the user to include an uppercase letter, a number, and a symbol
    - This ensures that the number of characters possible for a password increases, thus increasing password complexity.

Passwords were transmitted & stored as plain text

- In our initial prototype, the passwords were visible over the network and displayed in database as plain-text
- **Solution:**
  - Hash the passwords before being stored or transmitted
    - *SHA-256* hashing algorithm (*crypto-js*)

Known Security Vulnerabilities in dependencies

- Found 9 security vulnerabilities associated with npm (*nope package manager*)
  - Some versions of the packages are deprecated and not maintained
- **Solution:**
  - Fix by running this command in terminal:
    - ***npm audit fix--force***

URL Open Redirection Vulnerability

- The routing of the site allowed users to enter urls that they shouldn’t be allowed to access without being logged in.
- While they wouldn’t be able to tamper with the database through this, users still gained access when they shouldn’t
- **Solution**:
  - Disable this route for users that aren’t logged in.

Future Plans

- Use web-sockets in order to provide a better user experience when messaging, allowing messages to be direct
- Time out a user after too many failed login attempts
  - This is done by keeping track of login attempts in the backend and then issuing a timeout as a response from the server
- Implement session tokens and hashing of these tokens
  - This allows a better user experience for travelling between pages while keeping their logged in status even when revisiting the site after a period of inactivity

Lessons Learned

- Many times in class it has been mentioned that “as the number of features increase, the number of security risks increase”
  - This project was proof of this statement and just how impactful adding even a single feature can be on security
- No matter how secure our application/software is, there will always be vulnerabilities that we need to carefully consider

SecureDove

Thank you

Project Demo

Link to demo:
https://youtu.be/UUBG1-LMqJQ

Link to repository:
https://github.com/AliDzulfiqar/CptS428-ABFMJ-SecureDove