CS3354 Software Engineering
Final Project Deliverable 2

Scribble

Nicole Iwuala
Harini Arunprakash
Zaaim Rahman
Nicholas Donovan
Muhammad Ali
Amir Akilimali
Aditya Sinha
Pranav Balu

Table of Contents

Our Developers

Nicole Iwuala

I did a large portion of the non-functional requirements; revised and edited some of the functional requirements; made two of the sequence diagrams; made the project scope commit in GitHub; helped brainstorm use cases; and helped organize our shared Google Drive. For Deliverable 2, I calculated hardware and software costs. I helped create presentation content and design.

Harini Arunprakash

I worked on the use-case diagrams. For Deliverable 2, I worked on function point analysis, personnel cost, and LOC estimation.

Zaaim Rahman

I worked on constructing the functional requirements to specify our application's desired goals for our project and designing/constructing the class diagram's various classes and flow of information. For Deliverable 2, I helped on unit testing.

Nicholas Donovan

I created the project GitHub repository, revised and edited some of the non-functional requirements, created the architectural design with Muhammad, provided input for the class diagram, and drafted the Deliverable 1 report. I created the project test plan, wrote the unit test with Zaaim, assisted with the presentation, created the report for deliverable 2, and pushed the deliverable to the repository.

Muhammad Ali

I worked on the use-case diagrams and architectural design for the project. I worked on the comparison to similar projects and the presentation for deliverable 2.

Amir Akilimali

I worked on sequence diagrams and also worked on presentation slides for deliverable 2.

Aditya Sinha

I worked on the non-functional requirements and the class diagram. I did project scheduling for deliverable 2.

Pranav Balu

I worked on sequence diagrams and helped brainstorm ideas for implementation of the project. For Deliverable 2, I worked on the presentation slides.

Deliverable 1 Content

Overview

The motivation for tackling this project stems from the growing need for efficient communication in today's digital age. We recognize the importance of understanding technology in everyday life. Our goal is to provide a solution that not only facilitates communication but also enhances productivity, connectivity, and convenience in a world which requires instantaneous communication. The primary goal of our group is to build an instant messenger with a focus on accessibility, intuitive design, and simplicity. We envision our platform being used in various real-life scenarios, including personal communication, professional collaboration, and community building, catering to the diverse communication needs of individuals, teams, and organizations.

> *Good choice for a topic! Messages are an inherent part of our lives now and organizing them is becoming more and more of a requirement.*
>
> *It is great to see a detailed breakdown of the tasks you have worked on already. Good job.*
>
> *In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how.*
>
> *Fair delegation of tasks.*
>
> *Please share this feedback with your group members.*
>
> *You are good to go. Have fun with the project and hope everyone enjoys the collaboration.*

In the current collaborative landscape, messaging, and efficient communication are essential. In addition to our initial proposal, we have developed and incorporated a distinctive feature that sets us apart from other messaging platforms. We have all experienced the desire for better communication, feeling limited by text. We have overcome the issue by introducing a new feature that allows hand-drawn scribbles to be displayed over text, ushering in a new era of messaging.

Our Project

https://github.com/nick-donovan-utd/3345-TheSKidSociety

Our Methodology

We adopted the Agile methodology for our instant messenger. Agile enables our small team of developers with diverse specialties to concentrate on the software and the projects' vision, rather than extensively planning the design. Its iterative approach enables Scribble to become adaptable to changing requirements and environments while fostering collaboration within our team. Moreover, Agile enables us to enhance our messenger continuously by implementing incremental improvements, new features, and additions while preserving its heart.

Functional Requirements

The functional requirements of our software project lay out the services and capabilities our application will provide to end-users.

User Functionalities

Dynamic Communications

DCUF.1. The users of a session shall be able to draw, edit, and delete SVG-style graphics anywhere on the messaging session view, upload documents, and generate canvas spaces.

Messaging

MUF.1. The user shall be able to send, view, reply, and delete message components in sessions with various other users.

MUF.2. The user shall be able to generate sessions with multiple users.

Session Management

SMUF.1. The user shall be able to label, group, search, and delete previous sessions with previous users.

SMUF.2. The user shall be able to pin/favorite certain sessions with other users for quick access.

System Functionalities

Gestures

GSF.1. Message sessions shall have scrolling to view previous messages and expandable elements that can be tapped to zoom in/out.

Notifications

NSF.1. System shall notify the user when messages are received with or without drawn components.

Non-Functional Requirements

The non-functional requirements of our software project define the attributes and constraints that guide the behavior, ethics, and performance of our system.

Dependability

DE.1. The messaging system shall be available at all times except for short, infrequent scheduled maintenance times.

DE.2. The messaging system shall have an uptime of at least 99.5%.

DE.3 The messaging system shall be designed to be efficient and scalable, accommodating larger volumes of users and data over time without service degradation.

DE.4 The system shall utilize a distributed network architecture to ensure optimal availability.

Security

SE.1 The messaging system shall use encryption at rest and encryption in transit on message data and any other necessary architecture that will protect messages from unwanted access and modification.

SE.2 The messaging system shall be designed to protect against common security threats, including injections and data breaches.

SE.3 Users shall be authenticated such that only authorized users can send or receive messages.

Regulatory

RE.1 The organization shall comply with international and federal law specific to where services are available. The organization shall also endeavor to comply with state and local regulations to the best of its ability or where necessary.

RE.2 The messenger service shall comply with applicable data protection laws, including but not limited to, GDPR, CCPA, and other relevant regional or national regulations.

RE.3 The organization shall obtain explicit consent from users prior to collecting and processing their data in accordance with regulatory requirements in regard to user consent and privacy. Refusal to provide consent may result in the organization’s inability to provide service to the user.

RE.4 The organization shall fulfill required reporting mandates including disclosures related to data breaches and audits.

Ethical

ET.1 Messaging data shall be stored temporarily on the cloud for 1 month unless the user chooses to backup their data.

ET.2 The organization shall rely on user feedback from the platform to decide on future modifications to the system.

ET.3 The service and organization shall prioritize user privacy and data protection with strict confidentiality while adhering to regulatory obligations.

ET.4 The service shall not collect or store user data beyond what is absolutely required for essential functionality.

ET.5 User data shall not be shared unless: it is essential for the functionality of the service, in which case only necessary data required to provide the service will be shared, or when required to comply with a lawful order.

ET.6 The service shall have policies and mechanisms in place to prevent service abuse.

Usability

US.1 The messaging system shall offer multilingual support, including but not limited to English, Spanish, Chinese, and French..

US.2 The messaging system shall ensure accessibility for users with various disabilities including those with speech, visual, auditory, physical, and others, in accordance with applicable federal laws and regulations.

Environmental

EN.1 The messaging system shall be compatible with modern versions of common operating systems and environments, including but not limited to Windows, macOS, specific linux distributions, and mobile operating systems such as iOS and Android.

EN.2 The messaging system shall require internet connectivity in order to send or receive new messages.

Operational

OP.1 There shall be a schedule for routine maintenance, software updates, and security patches.

OP.2 The organization shall follow industry-standard monitoring and logging practices.

Development
DV.1 The organization’s development team shall adhere to industry standards for secure coding practices.

Performance
PE.1 Messages shall be delivered in real-time with minimal latency.

PE.2 The messaging system shall support 100,000 messages per second.

Space
SP.1 The user shall have an attachment size limit of 100 MB.

SP.2 The user shall have a message character limit of 4096.

SP.3 Messages shall be compressed and encrypted at rest on the users device, requiring the smallest storage footprint as possible.

SP.4 Cached photos, messages, and drawings can be limited to a size specified by the end user.

Accounting
AC.1 The organization shall maintain records of costs that involve development, testing, maintenance, and any other relevant project work.

AC.2 The organization shall release earning reports on a regular basis to provide transparency regarding its financial performance and motives. Releases shall comply with federal regulation and standards and be accessible to the public and to stakeholders.

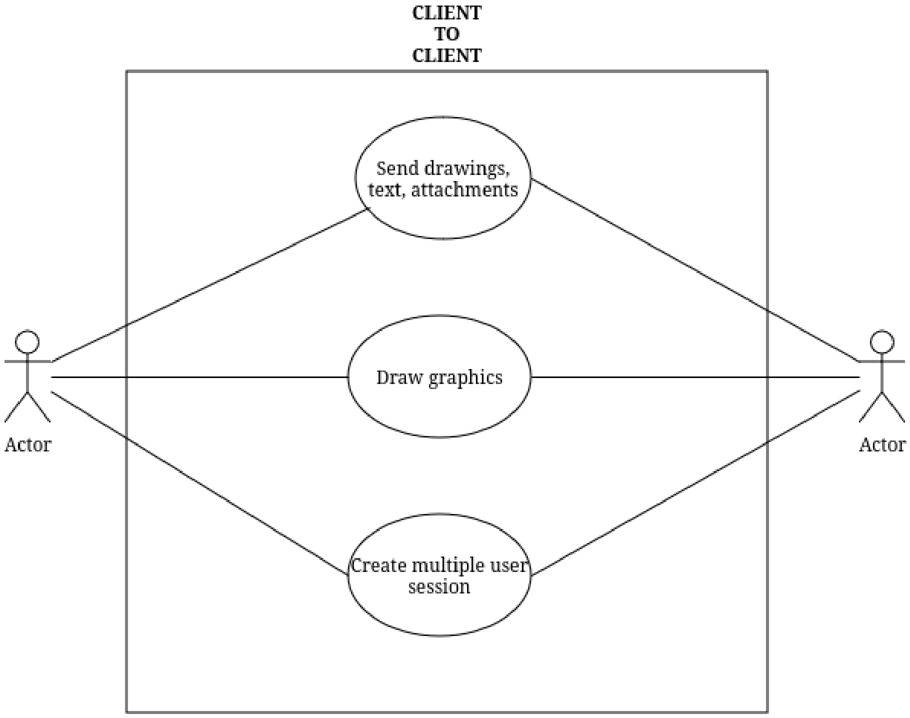
Safety/Security
SA.1 The organization shall conduct audits and vulnerability assessments and address issues during routine maintenance.

SA.2 There shall be access controls in place to define who can perform specific actions such as managing user accounts and pushing updates within the messaging system.

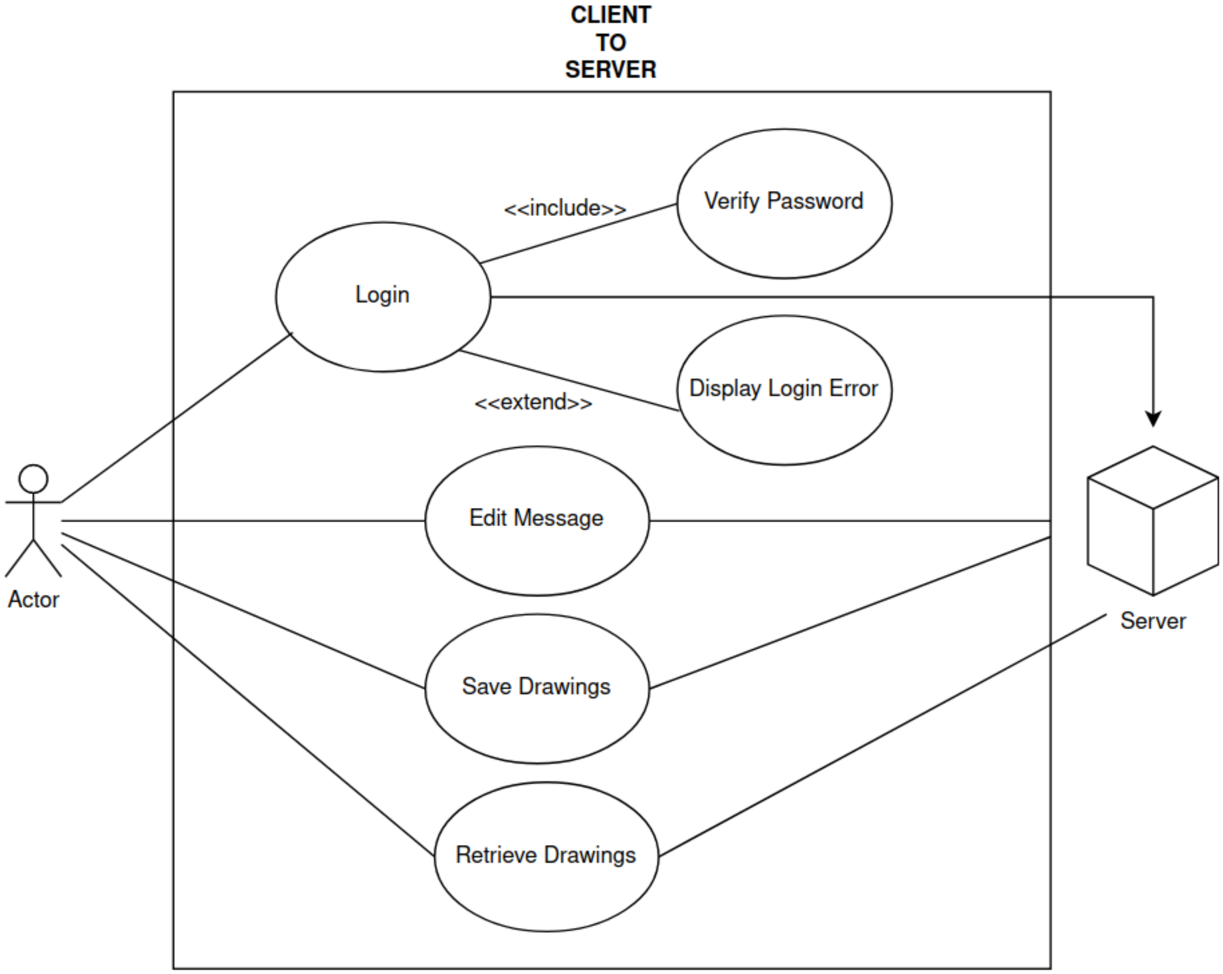
SA.3 The messaging system and services managed by the organization, shall undergo auditing and penetration testing at least every three years by a third-party organization. The details of the audit and penetration test shall be released to the public after any critical findings are addressed, up to three months after the findings are received.

Use-Case Diagrams

Client-to-Client

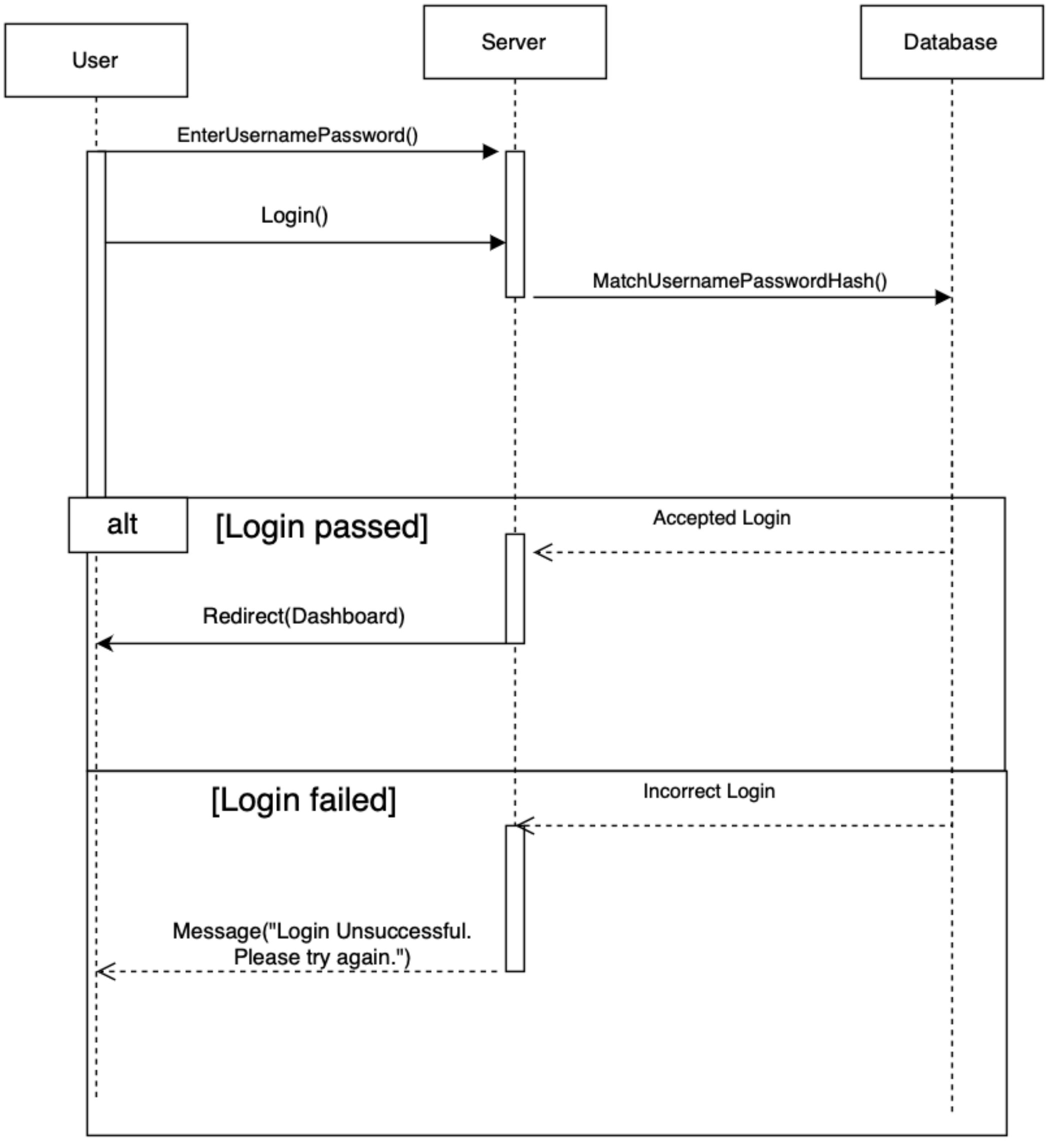

CLIENT
TO
CLIENT
Send drawings, text, attachments
Draw graphics
Create multiple user session
Actor
Actor

Client-to-Server

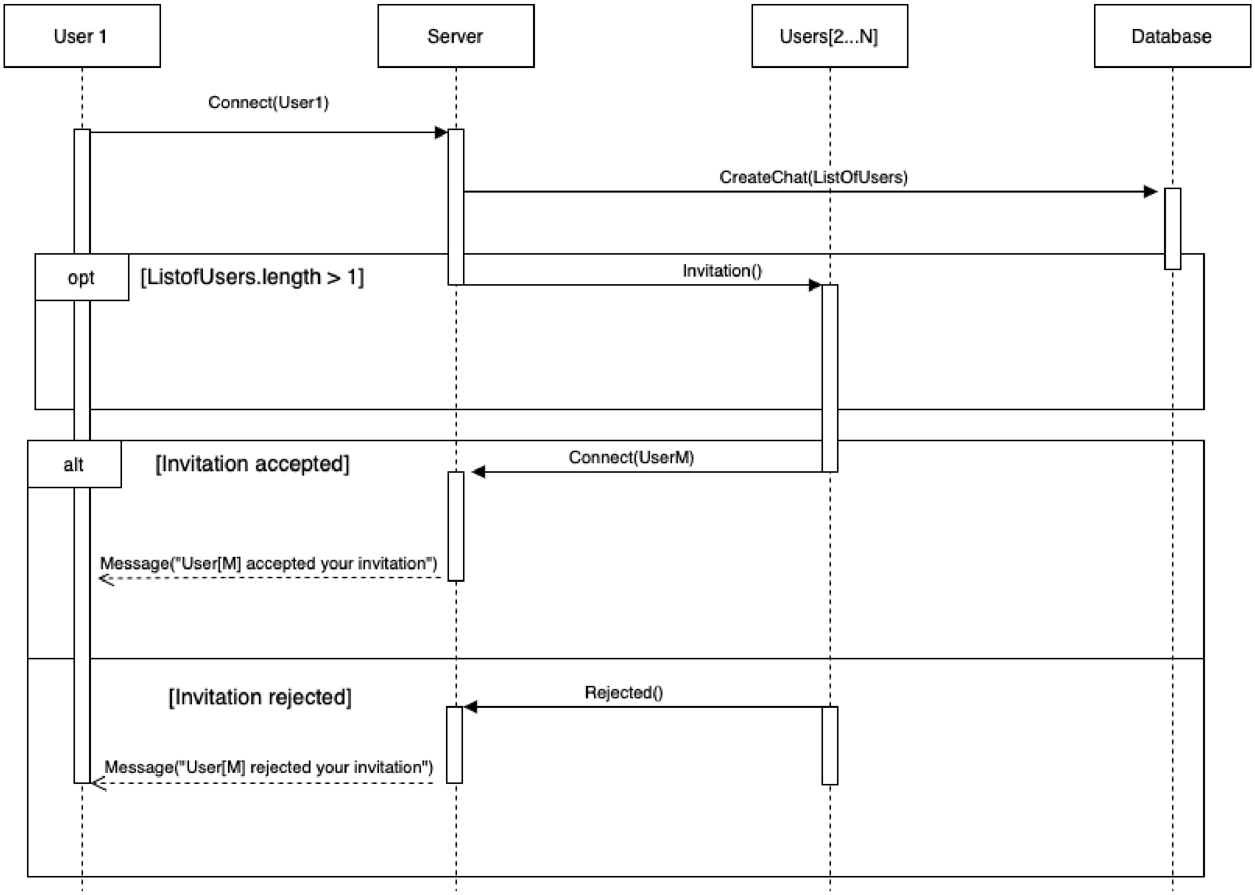


Sequence Diagrams

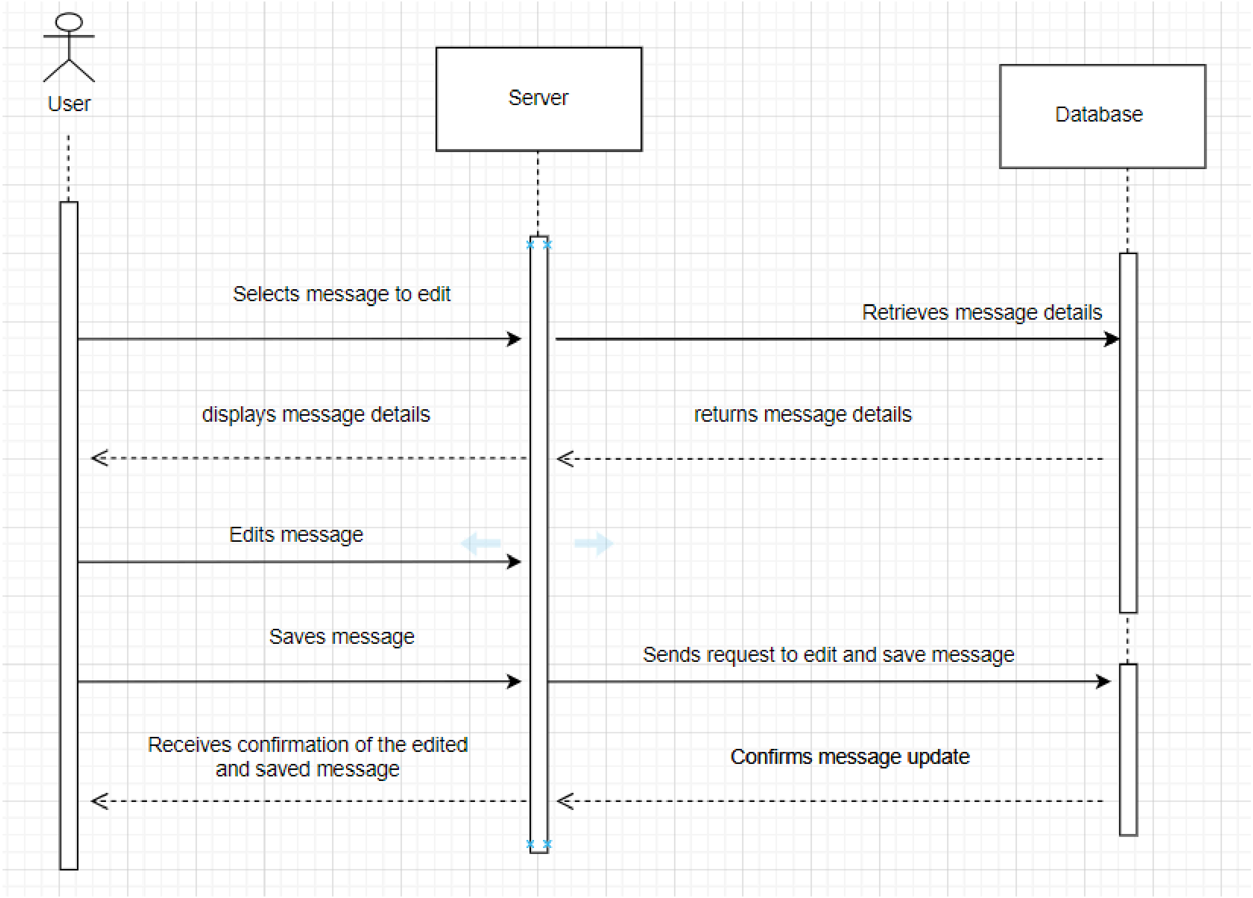
Login



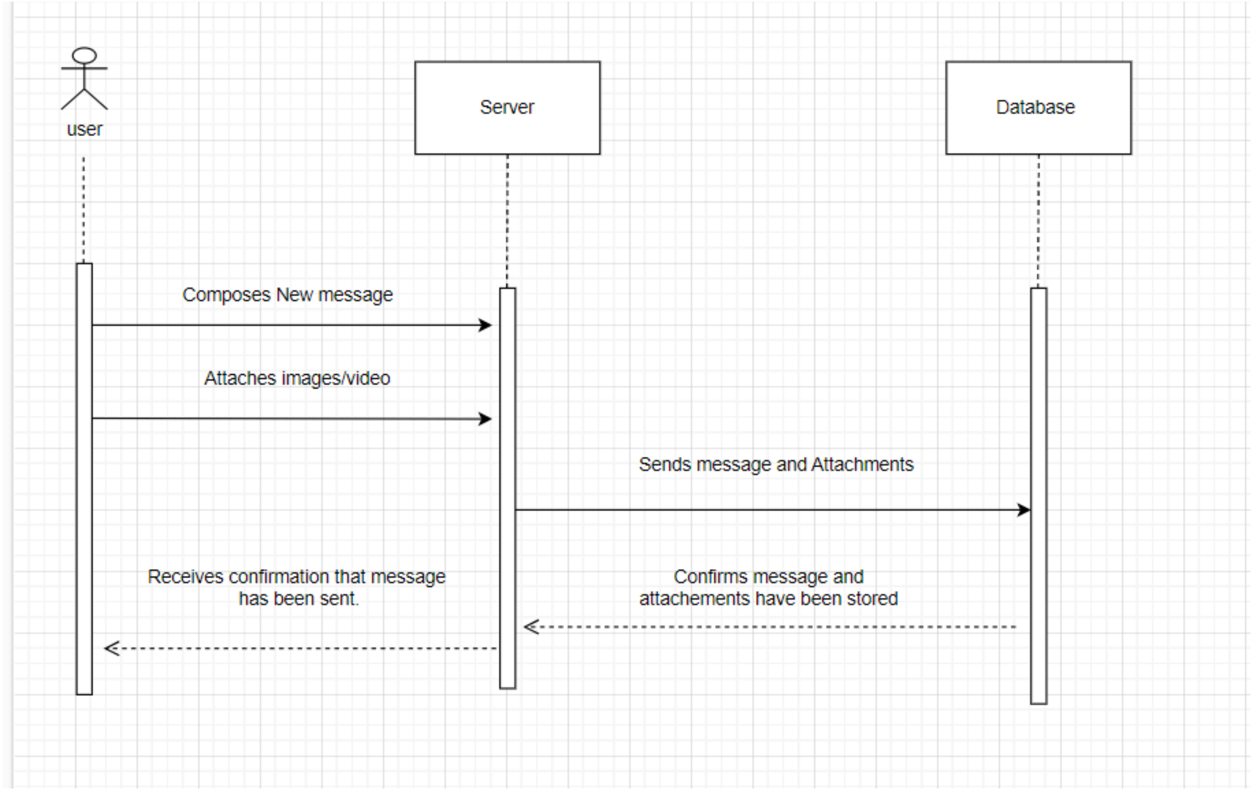
Create Group Session



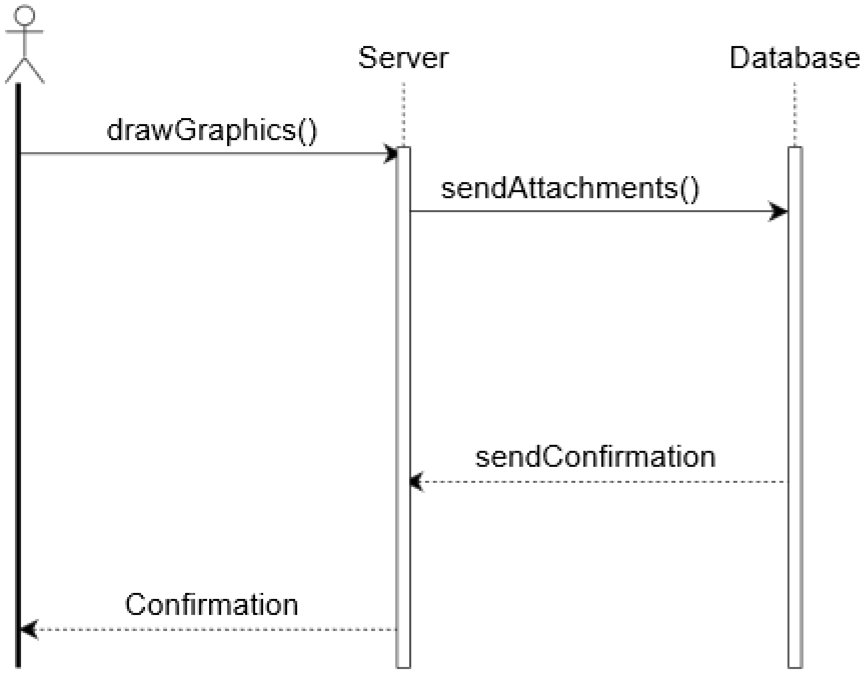
Edit Message



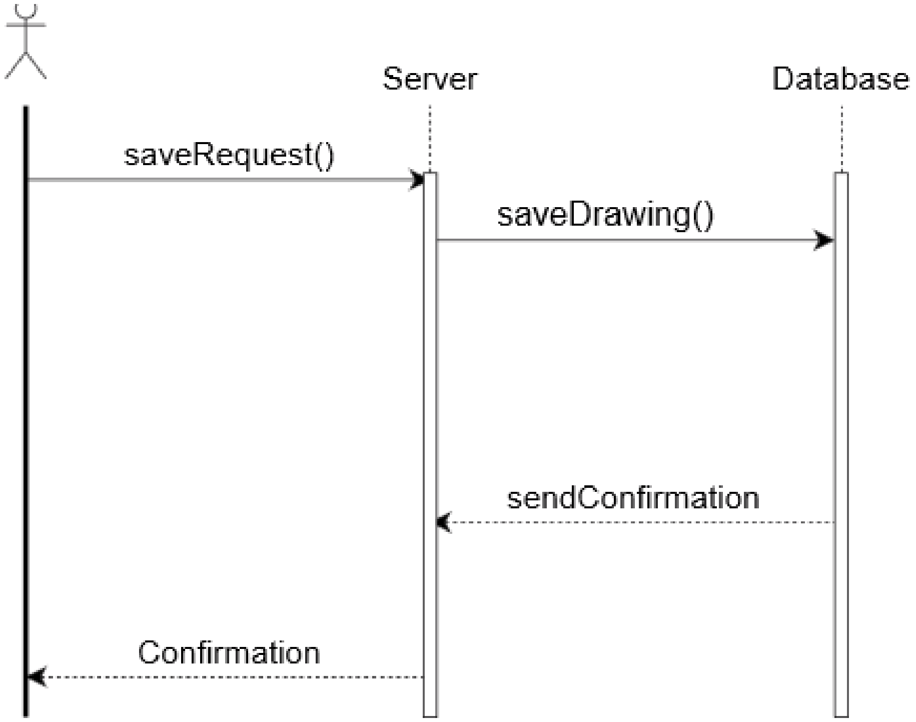
Send Message


user
Server
Database
Composes New message
Attaches images/video
Sends message and Attachments
Receives confirmation that message has been sent.
Confirms message and attachements have been stored

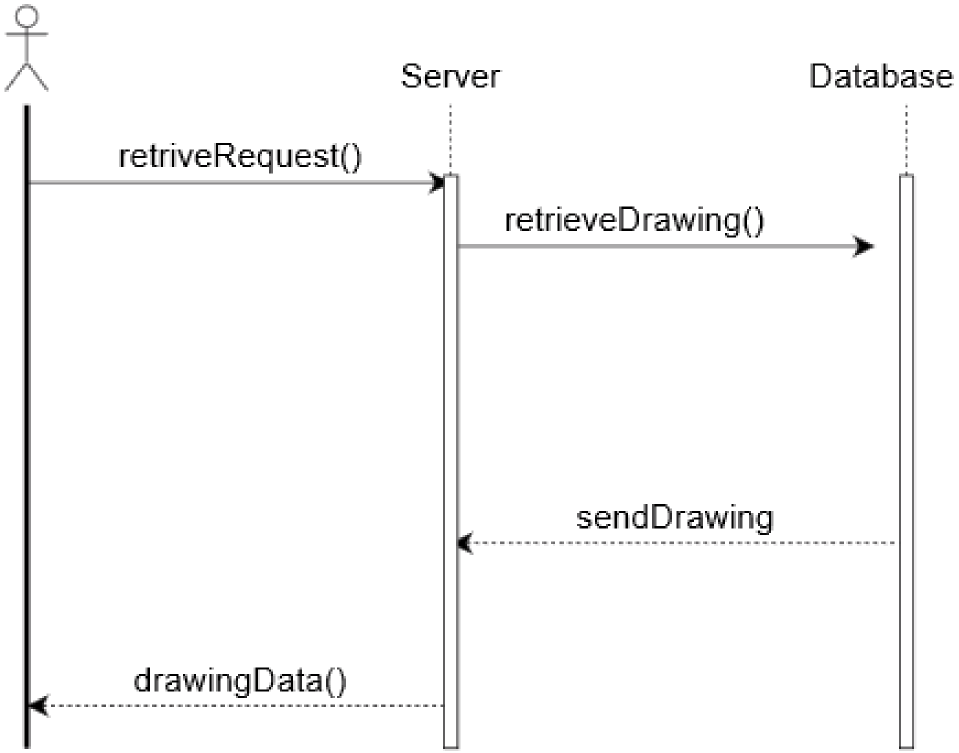
Draw



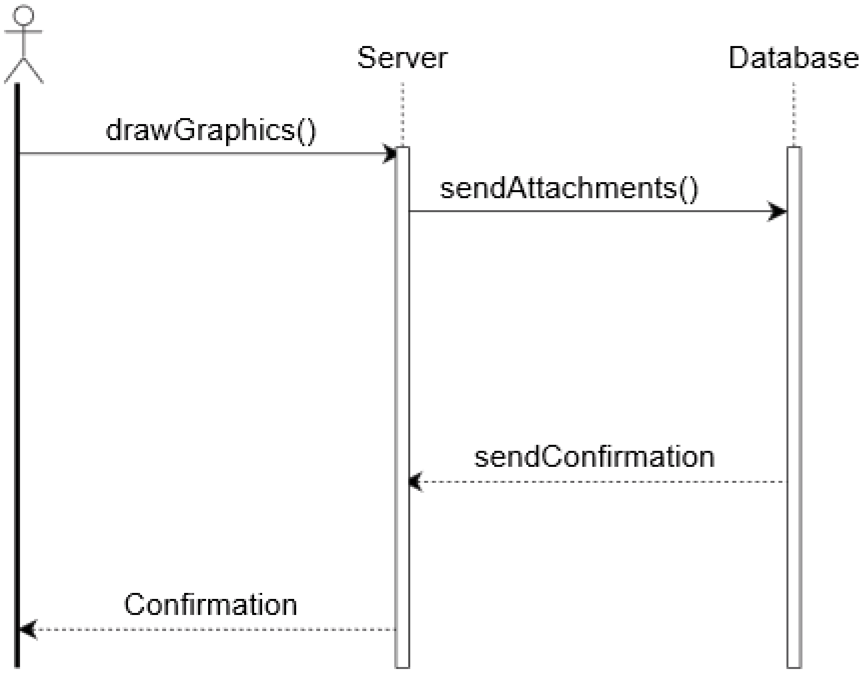
Save Drawings



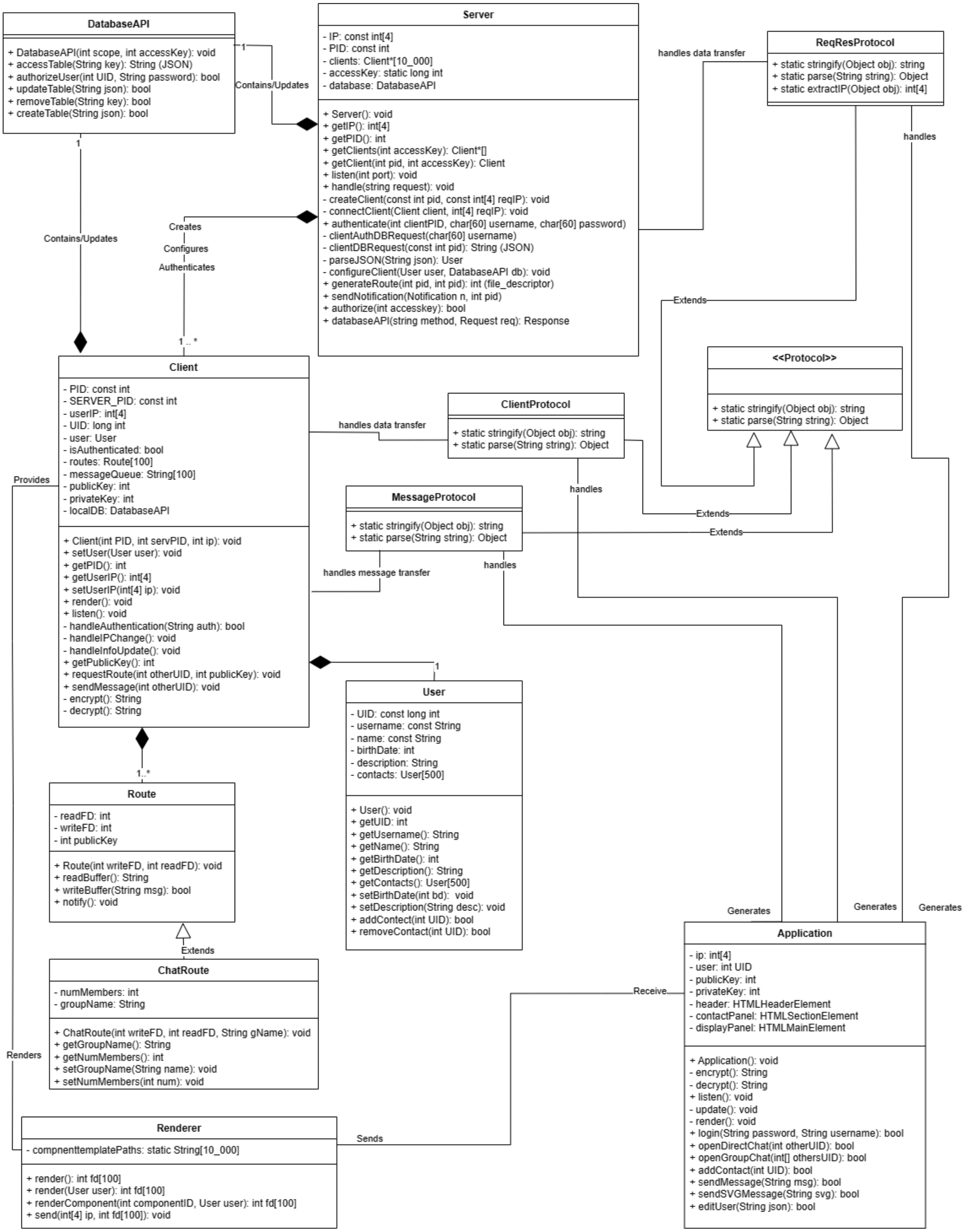
Retrieve Drawings



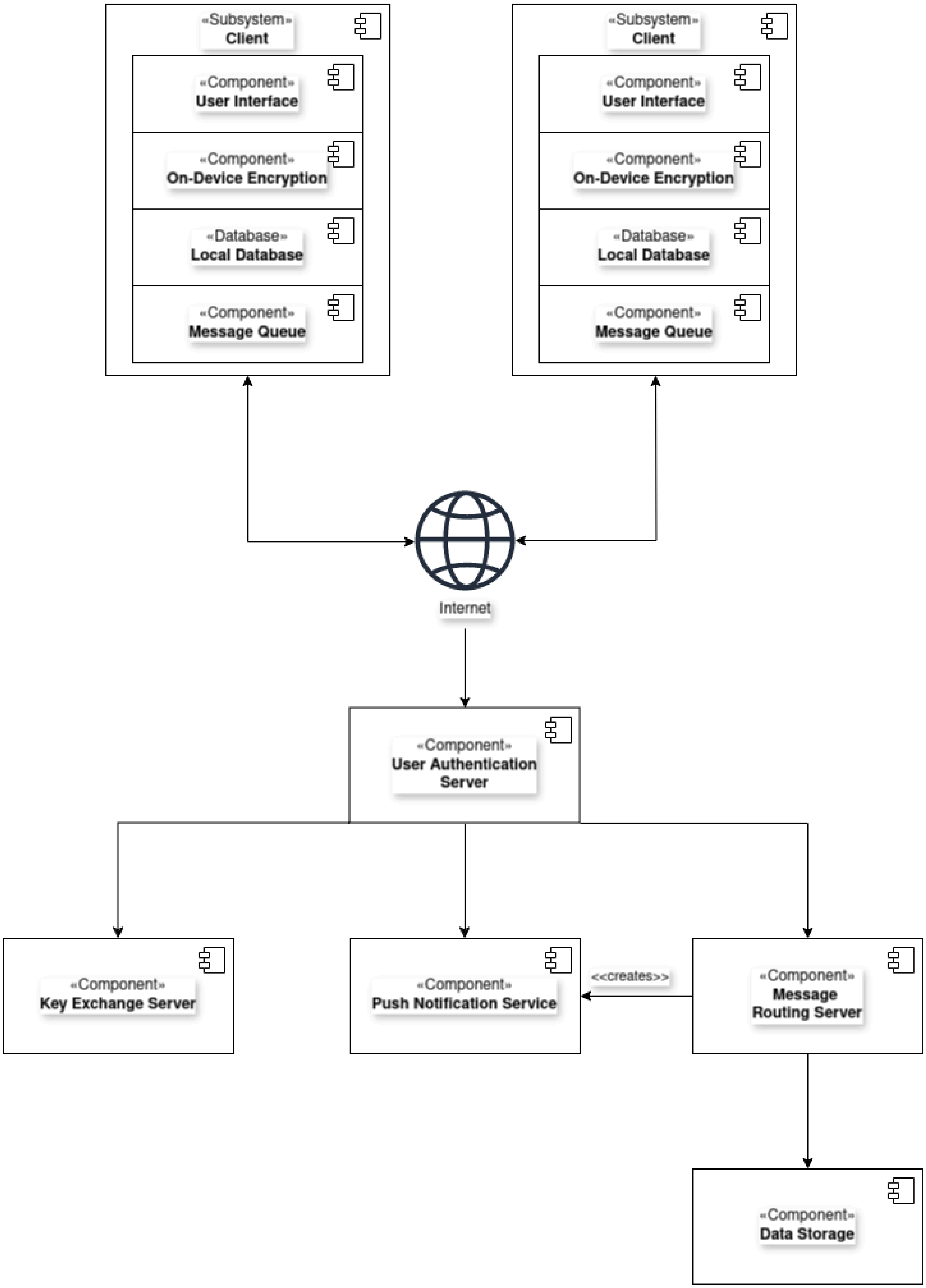
Draw Graphics



Class Diagram


DatabaseAPI
+ DatabaseAPI(int scope, int accessKey): void
+ accessTable(String key): String (JSON)
+ authorizeUser(int UID, String password): bool
+ updateTable(String json): bool
+ removeTable(String key): bool
+ createTable(String json): bool
1
Contains/Updates
Server
- IP: const int[4]
- PID: const int
- clients: Client*[10_000]
- accessKey: static long int
- database: DatabaseAPI
+ Server(): void
+ getIP(): int[4]
+ getPID(): int
+ getClients(int accessKey): Client*[]
+ getClient(int pid, int accessKey): Client
+ listen(int port): void
+ handle(string request): void
- createClient(const int pid, const int[4] reqIP): void
- connectClient(Client client, int[4] reqIP): void
+ authenticate(int clientPID, char[60] username, char[60] password)
- clientAuthDBRequest(char[60] username)
- clientDBRequest(const int pid): String (JSON)
- parseJSON(String json): User
- configureClient(User user, DatabaseAPI db): void
+ generateRoute(int pid, int pid): int (file_descriptor)
+ sendNotification(Notification n, int pid)
+ authorize(int accesskey): bool
+ databaseAPI(string method, Request req): Response
handles data transfer
ReqResProtocol
+ static stringify(Object obj): string
+ static parse(String string): Object
+ static extractIP(Object obj): int[4]
handles
1
Contains/Updates
Creates
Configures
Authenticates
1 .. *
Extends
Client
- PID: const int
- SERVER_PID: const int
- userIP: int[4]
- UID: long int
- user: User
- isAuthenticated: bool
- routes: Route[100]
- messageQueue: String[100]
- publicKey: int
- privateKey: int
- localDB: DatabaseAPI
+ Client(int PID, int servPID, int ip): void
+ setUser(User user): void
+ getPID(): int
+ getUserIP(): int[4]
+ setUserIP(int[4] ip): void
+ render(): void
+ listen(): void
- handleAuthentication(String auth): bool
- handleIPChange(): void
- handleInfoUpdate(): void
+ getPublicKey(): int
+ requestRoute(int otherUID, int publicKey): void
+ sendMessage(int otherUID): void
- encrypt(): String
- decrypt(): String
<<Protocol>>
+ static stringify(Object obj): string
+ static parse(String string): Object
handles data transfer
ClientProtocol
+ static stringify(Object obj): string
+ static parse(String string): Object
Provides
handles
Extends
MessageProtocol
+ static stringify(Object obj): string
+ static parse(String string): Object
Extends
handles message transfer
handles
1
User
- UID: const long int
- username: const String
- name: const String
- birthDate: int
- description: String
- contacts: User[500]
+ User(): void
+ getUID: int
+ getUsername(): String
+ getName(): String
+ getBirthDate(): int
+ getDescription(): String
+ getContacts(): User[500]
+ setBirthDate(int bd): void
+ setDescription(String desc): void
+ addContect(int UID): bool
+ removeContact(int UID): bool
1..*
Route
- readFD: int
- writeFD: int
- int publicKey
+ Route(int writeFD, int readFD): void
+ readBuffer(): String
+ writeBuffer(String msg): bool
+ notify(): void
Extends
ChatRoute
- numMembers: int
- groupName: String
+ ChatRoute(int writeFD, int readFD, String gName): void
+ getGroupName(): String
+ getNumMembers(): int
+ setGroupName(String name): void
+ setNumMembers(int num): void
Renders
Generates
Generates
Generates
Application
- ip: int[4]
- user: int UID
- publicKey: int
- privateKey: int
- header: HTMLHeaderElement
- contactPanel: HTMLSectionElement
- displayPanel: HTMLMainElement
+ Application(): void
- encrypt(): String
- decrypt(): String
+ listen(): void
- update(): void
- render(): void
+ login(String password, String username): bool
+ openDirectChat(int otherUID): bool
+ openGroupChat(int[] othersUID): bool
+ addContact(int UID): bool
+ sendMessage(String msg): bool
+ sendSVGMessage(String svg): bool
+ editUser(String json): bool
Receive
Renderer
- compnenttemplatePaths: static String[10_000]
+ render(): int fd[100]
+ render(User user): int fd[100]
+ renderComponent(int componentID, User user): int fd[100]
+ send(int[4] ip, int fd[100]): void
Sends

Architectural Design



Project Scheduling

Project Start Date: April 29, 2024 (Monday)
Project End Date: October 4, 2024 (Friday)

The dates mentioned assume a 40-hour work week, with an expectation of 8 hours of work per day such that weekends are not included in the schedule.

Given that the estimated effort for the project is 45 person-months and there are 8 team members, it can be estimated that each person-month corresponds to roughly 160 hours of work for a standard 4-week month. This gives a total of 7200 working hours (45*160), meaning 900 working hours per team member.

To calculate the project duration, we can divide the total working hours (7200) by the number of team members (8) and the number of hours worked per week (40). This calculation gives us a project duration of 22.5 weeks, which we round up to 23 weeks for simplicity.

Cost, Effort and Pricing Estimation

To calculate the estimated effort, we used the Function Point method of estimation and the Lines of Code experience-based estimation. While both methods were implemented, the Lines of Code estimation is more accurate and matches the project schedule.

Calculating the Gross Function Point:

Inputs:

- Login
- Search message
- Message
- Search contact
- Add contacts to group chat
- Add Files
- Drawing
- payment

User outputs:

- Message
- files/drawings
- Group chat
- User dashboard

User queries:

- Send messages
- Delete message
- Edit message
- Add contact to conversation
- Delete contact
- create chat
- Add people to chat
- Delete chat

Data files and relational tables:

- Users
- Groups
- Messages
- Media
- Businesses/universities

- Payment

External Interfaces:

- App
- API
- Hardware interfaces

	Function Category	Count	Simple	Average	Complex	Count × Complexity
1	Number of user input	8	3	4	6	24
2	Number of user output	4	4	5	7	16
3	Number of user queries	30	3	4	6	90
4	Number of data files and relational tables	6	7	10	15	42
5	Number of external interfaces	3	5	7	10	15

GFP = 24 + 16 + 90 + 42 + 15 = 187

Processing complexities:

(1) Does the system require reliable backup and recovery? - 4
(2) Are data communications required? - 5
(3) Are there distributed processing functions? - 3
(4) Is performance critical? - 4
(5) Will the system run in an existing, heavily utilized operational environment? - 2
(6) Does the system require online data entry? - 5

(7) Does the online data entry require the input transaction to be built over multiple screens or operations? - 1
(8) Are the master files updated online? - 4
(9) Are the inputs, outputs, files, or inquiries complex? - 2
(10) Is the internal processing complex? - 2
(11) Is the code designed to be reusable? - 3
(12) Are conversion and installation included in the design? - 0
(13) Is the system designed for multiple installations in different organizations? - 4
(14) Is the application designed to facilitate change and ease of use by the user? - 3

Processing complexity adjustment:

$PCA = 0.65 + 0.01\ (4 + 5 + 3 + 4 + 2 + 5 + 1 + 4 + 2 + 2 + 3 + 0 + 4 + 3) = 1.07$

Function Point:

$FP = GFP \times PCA = 187 \times 1.07 = 200.09$

Estimated effort (E) and project duration (D):

Assumed productivity: 18
Assumed team size = 8

$E = FP\ /\ productivity = 200.09/18 = 11.1161 \approx 12$ person-months
$D = E\ /\ team\ size = 12/8 \approx 2$ months

Lines of Code Estimation:

Database management - 14,800 LOC

- Media - 2400 LOC
- Message - 2800 LOC
- Contacts - 1600 LOC
- Groups - 1600 LOC
- Profiles - 2400 LOC
- Payment - 1600 LOC
- Chats - 1600 LOC
- Login - 800 LOC

Function	Estimated LOC

Database management (DBM)	14,800
API	1280
Application	4000
Web	8000

Total LOC = 28,080
Average productivity for systems of this type = 620 LOC/person-month (Given)
Labor rate = $8000 per month (assumed)

Based on the LOC estimate and the historical productivity data:

Cost per line of code = Labor rate / average productivity = $8000 / 620 = $13
Total estimated project cost = Cost per line of code × estimated LOC = $13 × 28,080 = $365,040
Estimated effort = estimated LOC / average productivity = 28,080 / 620 ≈ **45 person-months**

Hardware

IaaS & PaaS	Costs
AWS EC2 (using Amazon Linux OS) Storage-optimized 10 instances per month X2gd.2xlarge 8vCPU 128 GiB memory Elastic load balancing	$1,765.86 per month
AWS S3	$993.60 per month
AWS DynamoDB	$1768.33 per month
Total	$4527.79 per month

Software

SaaS/Software	Costs
AWS IAM	$0.00 per month
AWS Pinpoint	$100 per month
AWS CloudWatch	$482.50 per month
AWS Management Console	$0.00 per month
Third-party libraries	$500.00 per month
GitHub Enterprise	$21.00
Total:	$1103.50 per month

Cost of Personnel

A significant aspect we need to consider is the personnel cost, which directly impacts the project's budget and resource allocation.

Firstly, we have a team of 8 skilled software developers. To determine the annual cost per developer, we referred to Payscale [4], which suggests an average annual salary of $78,202 for software developers with 0-5 years of experience.

This figure, however, is only a part of the overall personnel cost. We must also account for benefits and taxes, which typically amount to around 30% of the base salary. So, for each developer, we added 30% of the salary to cover these additional expenses. For our team, this equates to $23,460.60 per developer annually.

So, the total annual cost per developer, including salary, benefits, and taxes, comes to $101,662.60.

Now we simply multiply the total cost per developer by the number of team members. Therefore, the total annual personnel cost for our software development team amounts to:

8 developers × $101,662.60 per developer = **$813,300.80** annually.

This calculation gives us a clear understanding of the personnel cost associated with our proposed software project, enabling us to effectively manage our budget and resources throughout the development process.

Our Test Plan

1. Summary

This portion of the proposal outlines the testing plan for Scribble. Proper testing is critical for the performance and functionality of our application and will instill confidence in our end-users that they can depend on our application to satisfy their needs. Through applying test-driven development principles and implementing appropriate tests, we will ensure the security, reliability, and expectations of our users.

2. Scope

2.1 New functions and features

All new functions and features that are public-facing immediately or indirectly are within the scope of testing.

2.2 Performance testing

Performance testing on all supported environments to ensure responsiveness, scalability, and quality of service are within the scope of testing.

2.3 Security testing

Security testing on all public and indirectly public modules to ensure data protection is within the scope of testing.

2.4 Compatibility testing

Compatibility testing across supported environments on all public and indirectly public modules is within the scope of testing.

2.5 Usability testing

Usability testing on all public and indirect public modules to ensure an innovative and seamless interface is within the scope of testing.

3. Schedule

3.1. Unit Testing

Ongoing throughout the software lifecycle.

3.2. Component Testing

Parallel to unit testing.

3.3. System Testing

Conducted after integration of all modules before any major release.

3.4. Regression Testing

Conducted after each update and change to the software.

3.5. Performance Testing

Conducted after the final stages of development and before major releases.

3.6. Integration Testing

Concurrent with performance testing.

3.7. Security Testing

Ongoing throughout the software lifecycle.

3.8. Beta Testing

Scheduled before major releases for selected candidates.

4. Development Testing

Various tests will be conducted during the development process to identify bugs and defects early. The testing will be carried out in three stages: unit testing, component testing, and system testing.

4.1. Unit Testing

Guideline-based unit testing practices will be followed for formatting and selecting unit tests. This will streamline the testing process while ensuring that critical aspects of the code are covered. Unit tests will be written for individual components such as message sending, receiving, user authentication, and contact management.

4.2. Component Testing

Component testing will focus on testing individual modules or components of the application such as user interface components, database interactions, and message encryption/decryption.

4.3. System Testing

System testing will verify the integrated system against our requirements, allowing functional, performance, and compatibility testing across several of the defined environments specified.

4.4. Test Driven Development

Test-Driven Development will be utilized such that tests are written before any code is implemented ensuring that the code meets requirements from the start.

5. Release Testing

Testing will be conducted before any release of the application to end users.

5.1. Regression Testing

Regression testing will be used to ensure that new updates and changes remain backward compatible and do not affect the existing features of the application unless it is imperative for further development.

5.2. Performance Testing

Performance testing will assess the responsiveness and scalability of the application for supported environments and varying load conditions. Performance testing will include stress testing, load testing, and endurance testing.

5.3. Integration Testing

Integration testing will help verify and collate the different modules of the application to ensure a seamless and intuitive system.

5.4. Security Testing

Security Testing will be performed to identify vulnerabilities and ensure that the application is resistant to unauthorized access, data breaches, and other security threats to the best of our ability and in compliance with non-functional requirements SE.2 and SE.3.

6. User Testing

User testing will involve real end-users to evaluate and provide feedback on the usability, functionality, and experience of the application.

6.1 Beta Testing Service

The application will be released to a limited group of beta testers who will provide feedback before major releases. This group will help identify usability issues, bugs, and areas for improvement that need attention before public release.

7. Conclusion

The following test plan outlines the approach for Scribble, which covers the development, release, and user testing phases. All of these phases contribute towards creating a reliable application that our users can trust. This aligns with our mission of providing high-quality, reliable, and secure instant communication to our users.

Code for the Unit

(Full source files are available on GitHub)

```
package scribble.controller;

import scribble.exception.UserNotFoundException;
import scribble.dto.NewUserRequest;
import scribble.models.User;
import scribble.models.UserLogin;
import scribble.repository.UserLoginRepository;
import scribble.repository.UserRepository;
import jakarta.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;


@RestController
public class UserController {
  @Autowired
  UserRepository userRepository;

  @Autowired
  UserLoginRepository userLoginRepository;

  /**
   * Creates a new user with the provided details.
   *
   * @param newUserRequest The request object containing details for the new user.
   * @return The newly created user entity.
   */
  @PostMapping("/user/add")
  public User createUser(@Valid @RequestBody NewUserRequest newUserRequest) {

      // String firstName, String lastName, String email
      User user = new User(newUserRequest.getFirstName(), newUserRequest.getLastName(),
newUserRequest.getEmail());

      // Save login information
      UserLogin login = new UserLogin(user, newUserRequest.getUsername(),
newUserRequest.getPasswordHash());
      userRepository.save(user);
      userLoginRepository.save(login);

      return user;
  }
}
```

Test for the Unit

(Full source files are available on GitHub)

The following tests use MockMvc [3]. We are testing the createUser function to ensure that the user data is properly added to the server's database and returned.

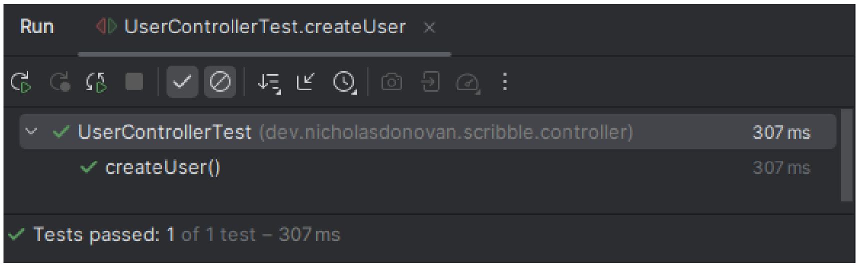
```
// Imports removed for space, available on GitHub
@SpringBootTest
@AutoConfigureMockMvc
class UserControllerTest {
  @Autowired
  private MockMvc mockMvc;
  private final String expectedFirst = "John";
  private final String expectedLast = "Smith";

  // Generate random email and username to prevent conflicts caused by uniqueness constraints.
  private final String randomEmail = java.util.UUID.randomUUID() + "@test.com";
  private final String randomUsername = java.util.UUID.randomUUID().toString();
  @Test
  void createUser() {
     // Define a JSON string for the request body.
     String newUserJson = """
           {
              "firstName": "%s",
              "lastName": "%s",
              "email": "%s",
              "username": "%s",
              "passwordHash": "$argon2i$v=19$m=65536,t=2,p=4$c29tZXNhbHQ$FgNukdXUKj/gS6Ur+fgQ6laMZLTrvSKo"
           }
           """.formatted(expectedFirst, expectedLast, randomEmail, randomUsername);

     // Perform a POST request to the '/user/add' endpoint using mockMvc for testing
     String result = assertDoesNotThrow(() -> mockMvc.perform(MockMvcRequestBuilders
           .post("/user/add")
           // Set the 'Content-Type' header to json and send the data
           .contentType(MediaType.APPLICATION_JSON)
           .content(newUserJson))
     .andReturn().getResponse().getContentAsString());

     // Test the results
     assertDoesNotThrow(() -> testResponseHelper(result));
  }
  private void testResponseHelper(String response) throws JSONException {
     // Parse and test the data
     JSONObject r = new JSONObject(response);
     assertEquals(expectedFirst, r.getString("firstName"), "TESTING: user creation (firstName)");
     assertEquals(expectedLast, r.getString("lastName"), "TESTING: user creation (firstName)");
     assertEquals(randomEmail, r.getString("email"), "TESTING: user creation (firstName)");
  }
}
```

The test result



```
✓ UserControllerTest (dev.nicholasdonovan.scribble.controller)   307 ms
   ✓ createUser()                                                 307 ms

✓ Tests passed: 1 of 1 test – 307 ms
```

Comparison to Competitors

Comparing our app with three competitors.

Snapchat

Snapchat has been a pioneer in messaging and multimedia sharing, appealing to a broad user base through features like self-destructing messages, stories, filters, and lenses. Its focus on instant visual communication has made it popular among younger demographics who favor quick, spontaneous interactions [1].

NoteIt

NoteIt positions itself as an intimate form of social networking, emphasizing closer connections by allowing users to share thoughts and moments directly on the home screen. NoteIt seems to aim for more persistent and meaningful engagements Its widget-centric approach suggests an always-on portal into friends' lives, potentially offering a quieter alternative to the noisy, algorithm-driven experiences of larger platforms [2].

PictoSwap

PictoSwap is designed for artistic expression and sharing, promoting a creative community where drawings are not only shared but experienced in the making. It serves users who take pleasure in the creative process and enjoy the interactive aspect of art creation. PictoSwap's real-time drawing feature, which allows friends to watch as a drawing is made, offers a unique, performative dimension to art sharing, which could be especially appealing to those who appreciate the journey of creation as much as the final artwork.

Our Offering

Our app combines all three of these features and creating something new. Snapchat is used more for communication and groups which our app will allow. NoteIt is an app that focuses more on sharing moments directly with someone which our app will allow through messaging and being able to send drawings. PictoSwap focuses more on artists and people who are interested in watching the progression of artwork. Our app uses scalable vector graphics which could appeal to both casual doodlers and professional designers, offering precision and quality that bitmap images lack. The group chat functionality suggests a collaborative angle, where ideas and creations can be discussed and developed in a community setting.

Conclusion

Scribble is an innovative messaging application that strives to improve communication in today's digital era. Our team has crafted a comprehensive solution that offers an unparalleled user experience. We've dedicated significant time and resources to meticulously plan and design every facet, ensuring that it exceeds expectations in both quality and performance. As you've seen, our roadmap encompasses all vital components of the application, and we've paid close attention to every detail to ensure seamless functionality.

Throughout the process of documenting our proposal, we have adjusted our vision to ensure our product is unique and new to the market. Additionally, we adjusted our use–case and sequence diagrams to better fit the requirements provided to ensure clarity.

Scribble is dedicated to meeting the needs of our users. To achieve this, we have developed an intuitive interface that is both user-friendly and loaded with robust capabilities all while upholding the highest standards of transparency and accountability, allowing us to remain financially sustainable without compromising on quality.

Our platform is meticulously designed with accessibility, intuitive design, and simplicity as top priorities. Our objective is to offer a flexible solution that meets the diverse communication needs of individuals, teams, and organizations. With applications ranging from personal communication to professional collaboration and community building, our platform strives to revolutionize the messaging experience.

Join us on this transformative journey as we embark on revolutionizing communication in the digital age with Scribble.

Citations

[1] “Snap Inc. - About Snap,” investor.snap.com. https://investor.snap.com/about-snap/

[2] “What Is NoteIt App & How Do You Use It?,” ScreenRant, Feb. 01, 2022. https://screenrant.com/what-is-noteit-app-how-use-explained/

[3] “Getting Started | Testing the Web Layer,” Getting Started | Testing the Web Layer. https://spring.io/guides/gs/testing-web

[4] “Software Developer,” Payscale.com, 2012. https://www.payscale.com/research/US/Job=Software_Developer/Salary