**CS673S16 Software Engineering** 

**Team 4 - Project Name**

**Software Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Daniel Budris | Team Lead, Configuration Lead |  |  |
| Shaurya Muttreja | Design Leader, Requirements Leader | *SM* | 11/07/2017 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
|  |  |  |  |
|  |  |  |  |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_buttcq9i221r)

[Design Patterns](#_x18fj36s1121)

[Key Algorithms](#_mtfbusfb0eq3)

[Classes and Methods](#_7ucksmkf6rzx)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Introduction

This document is intended to provide an overview of the architecture of the features we are implementing to extend 3blueprints, as well as document the overall functioning of the communications tool and how it relates to the larger system. It is also intended to document and explain the reasoning behind the frameworks we are using for this purpose.

# Software Architecture

**Class Diagram for system:**

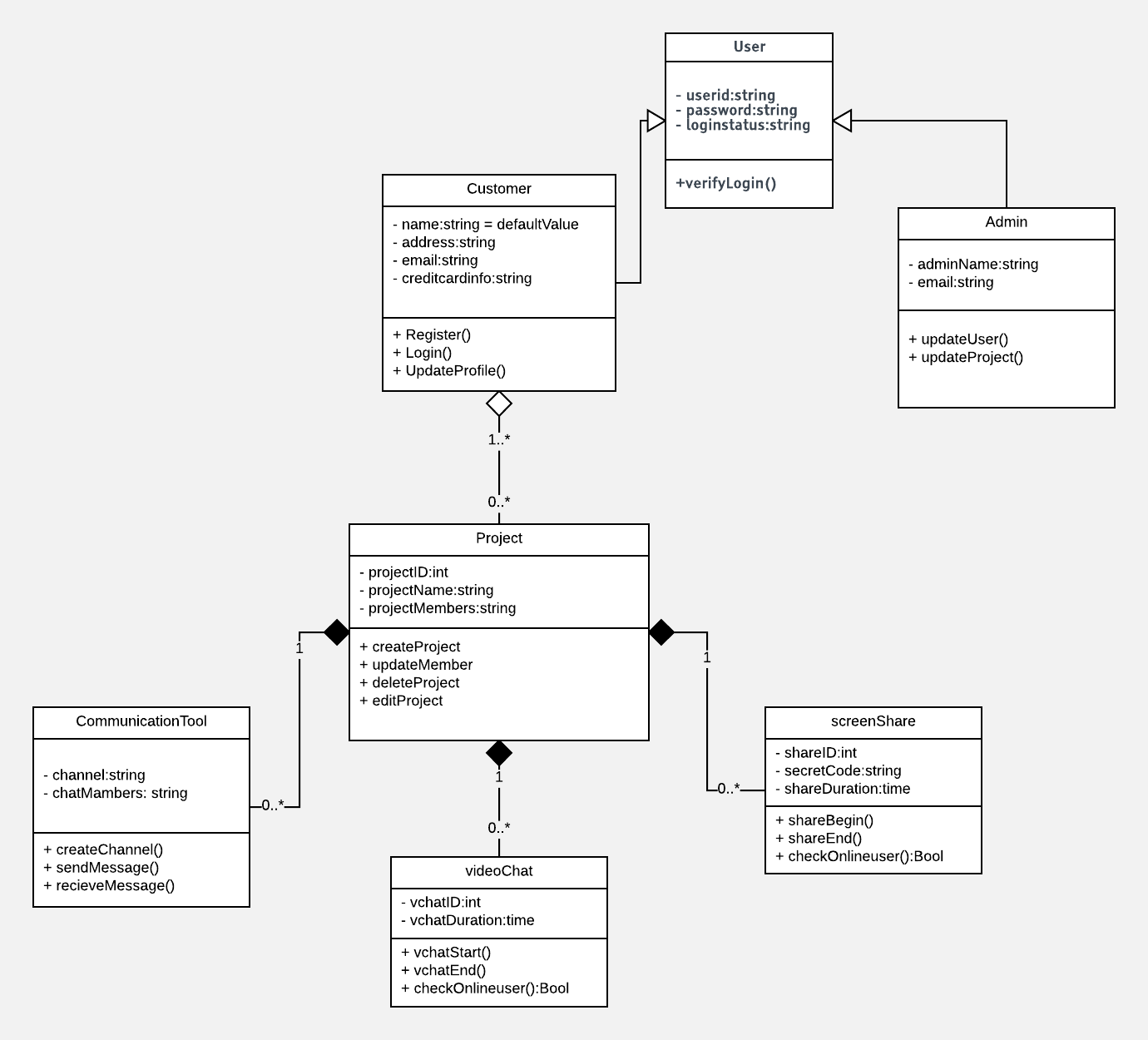


Figure 1.1 is the class diagram for the system in use. It displays the structure of the system by showing the classes present, the attributes, operation and the relationship among the attributes

**Communications tool chat features class diagram:**

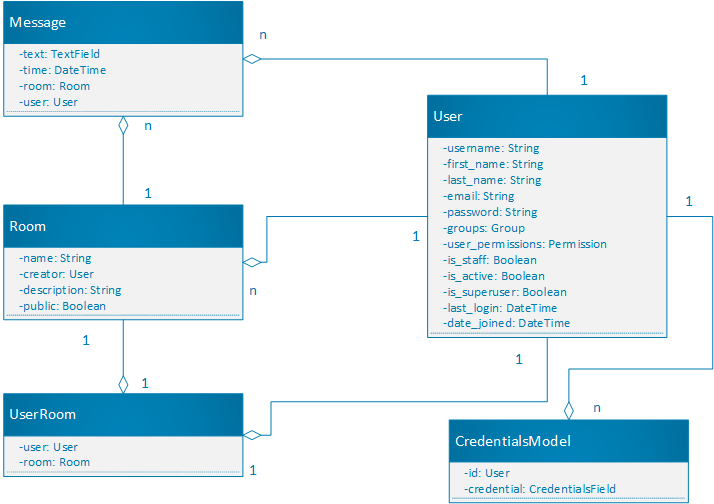


Figure 1.1 is class diagram represents the current state of the chat feature in the communication tool. It represents the state of the application architecture at the start of this project before modifications were made. As we start implementing more features and make significant updates to the system we will be updating this class diagram

**Communications tool video chat class diagram:**

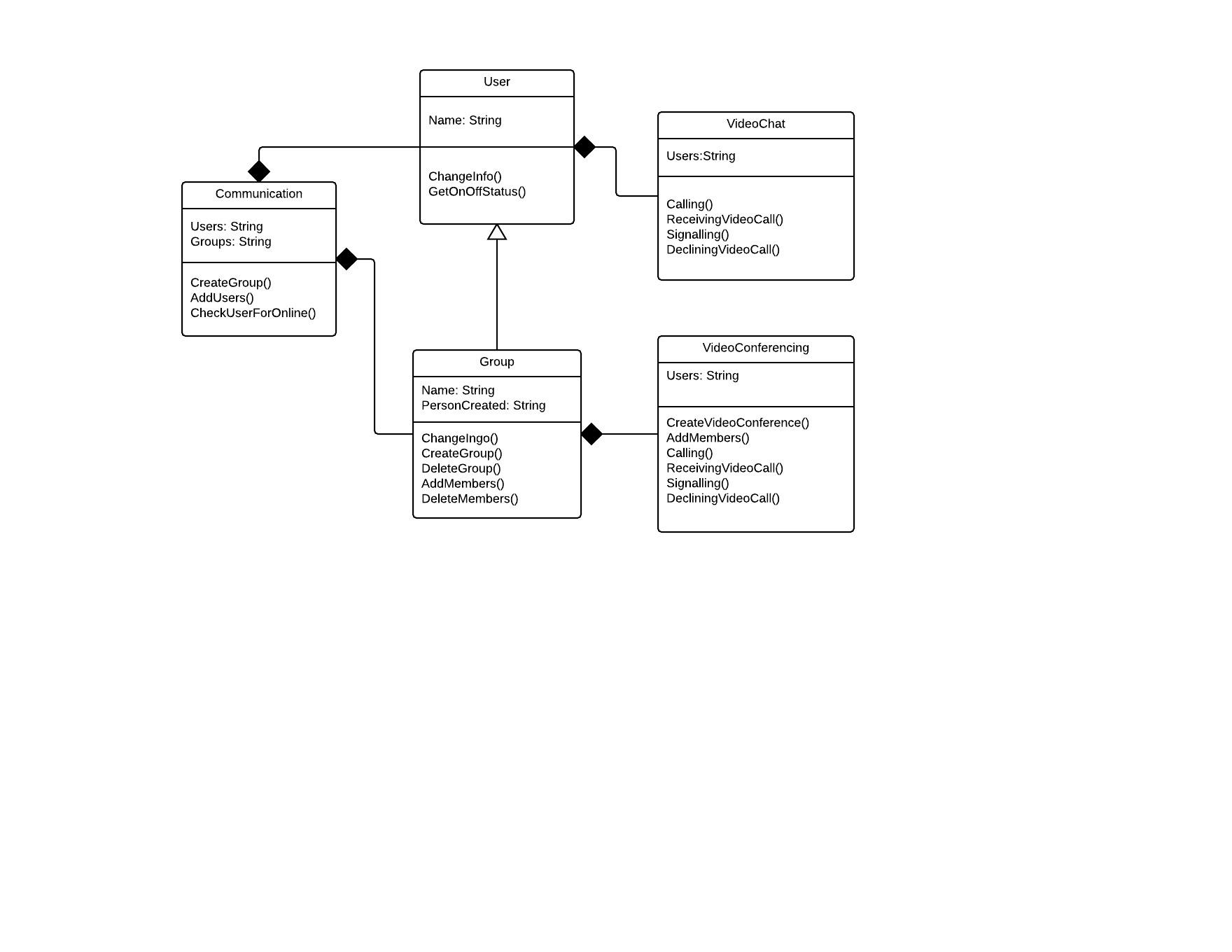


Figure 1.2 is a class diagram of the video chat application, which is being integrated into the existing communications tool. It displays the relationships between the users, groups and video chat entities and how they are composed.

There are User, Room, Message, and UserRooms classes currently represented in the Communication Tool. The User class has a one-to-many relationship with the Message and Room (as creator) classes. The Room class has a one-to-many relationship with the Message class. The UserRoom class has a one-to-one relationship with both the User and Room classes. With the NodeJS backend, users, rooms, and messages are stored in memory. Multiple users can use the same room and create multiple messages. User, room, or message information is stored persistently from the NodeJS backend by making REST API calls to the Django backend. We will be adding a CredentialsModel class to the other Django classes. This class is associated with the Google Drive API and will used to authenticate users.

**Use case diagram for the system:**

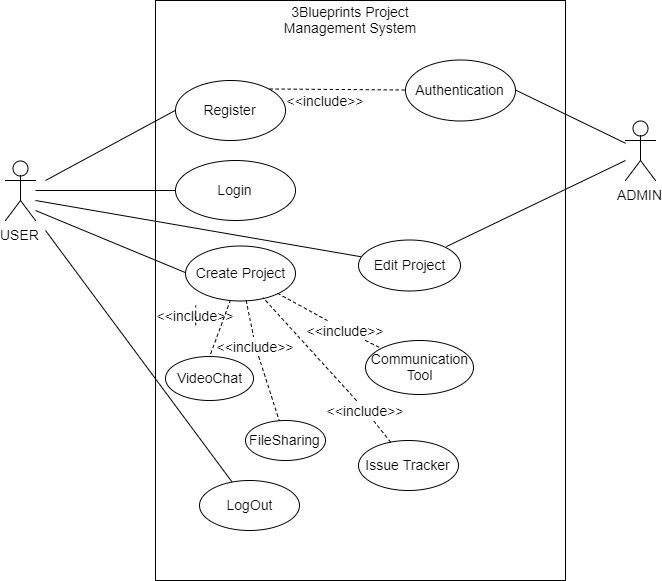
****

Figure 1.3 is a use case diagram, which represents the interactions among the elements of the system. This is the general use case diagram used for conceptual level explanation.

The application currently implements, and we are expanding on, four major frameworks:

[Django](https://www.djangoproject.com/) (Python web framework)

[Node.js](https://nodejs.org/en/) (Javascript backend platform)

[Express.js](https://expressjs.com/) (Node JS framework for building simplified RESTful APIs and servers)

[Socket.io](https://socket.io/) (Node JS framework for bi-directional real-time communication)

Django is python web framework that enables Model-View-Controller architecture and forms the backbone of this application. Django is also responsible for user management.

Node.js is a Javascript platform which enables use of asynchronous, non-block code to be written on the server side. Currently the chat application uses node.js for the peer-to-peer communication API, through the implementation of Express.js.

Socket.io is a javascript framework which enables bi-directional real-time communication

WebRTC is a real-time and enables

# Design Patterns

* + The application is built primarily using the MVC design pattern, as structured by the implementation of Django.

# Key Algorithms

As we move more deeply into the video chat application we will update this section with any key algorithms we encounter or implement. It is still too early at this point to choose specific key algorithms.

# Classes and Methods

Automated class definition tools are still under consideration. Right now the primary classes are represented in the above class diagrams.

# References

# Glossary