Linnéuniversitetet
Kalmar Växjö

Final Report

RedRiver Chat

Authors: Jimmy Bengtsson, Andrew Galbraith, Sofia Kristiansen, Linda Ott Olander
Tutor: Tobias Ohlsson
Semester: VT18
Course: 1DV611
Date: June 4th 2018

Abstract

This report documents a software development project conducted within the scope of course 1DV611 Software Development in a Group at Linnaeus University. The project goals were decided upon in conjunction with an external company, RedRiver, a software consulting company in the field of .NET and C#. RedRiver assumed the role of customer to our developer group, and as such provided feedback during the iterative development process.

The software in question was a chat application capable of text chat (including images) and video chat (including live streaming) in the fashion of existing apps such as Skype, Twitch or Slack. Initial requirements were extensive; among these were the existence of a user, admin and super user hierarchy, stringent security and encryption measures and a focus on usability.

Development followed an agile/UP methodology and the given timeframe was 10 weeks, divided into inception, elaboration, construction and transition stages. It was apparent from the outset that the allocated project time would not permit full requirement implementation, and this was accommodated into the project's time planning. Development began with a careful requirements analysis and a number of proof of concepts to reduce risk. Architecturally the project consisted of a React client, an ASP.NET Core server and an SQL database, all hosted in a Microsoft Azure cloud environment. The SignalR library was used to facilitate chat functionality, primarily over websockets but with a long polling fallback. SendGrid was used to send out account verification emails to newly registered users.

Despite varying requirements from the customer during development, the project was judged to be successful by all group members, and there were no major setbacks or significant technical problems. A fully working app was developed with text and limited video chat functionality. Although far from all requirements were implemented, the project provides a stable base for further development.

Table of contents

Introduction/Background

This project was part of the Linneaus University course 1DV611 - Software Development Project in a Group - and was conducted in association with RedRiver, a company providing consulting in .NET-framework and C# projects. RedRiver offer customers help with web development, system development and project management in web and system development projects.

RedRiver saw a need for an easy-to-use and secure application for smartphones and tablets which could facilitate text and image chat between two or more users, as well as live video calls. They wished the graphical user interface to be easy to understand and use, since the application was targeted towards a user group where some may have little or no prior experience with IT and chat applications.

The application was also to be suitable as an internal communication solution for RedRiver's clients.

Our group consisted of four members, all of whom were distance students during the course of the project. Our resources in terms of time to work on the application were 180 hours per person over the span of ten weeks.

Purpose and goals

The purpose of the project was to work with a real world customer and to solve the problem which they provided i.e. the development of a chat application for mobile and tablet devices. As the project was to be conducted in a group, the purpose was also to gain experience in discussing different types of solutions to the problem and to use knowledge gained throughout our studies in practice.

The application was aimed at RedRiver's clients in the public sector and larger corporations that had a need for a communication application with high security and accessibility requirements. The application was primarily intended for use as an internal communication solution for RedRiver's customers. Users with little prior experience of dealing with computers or mobile devices were also to be able to use the application without problems.

The application was to be a foundation that RedRiver could continue to build on and further develop outside of this project.

Overall goals

The course goals for the students were to:

- follow a standardised model for software development and conduct a project in a group.
- write and continuously update the project documentation.
- critically examine other projects and their documentation.
- present the groups method and results both in written and oral form in a way adapted to the user group.

The project's overall goals were to:

- continuously throughout the project interact with the customer.
- continuously make deliveries of the system to the customer.
- in iterations develop and deliver an application according to the customer RedRiver's requirements.

Base features

1. The application should work on smartphones and tablets.
2. The GUI should be easy to understand and use (designed according to WCAG 2.1, level AA).
3. Users should be able to create chat rooms to send text messages and images to each other. Users should also be able to make video calls to each other.
4. Users should be able to add other users as friends.
5. The entire system (i.e. client, server, storage and data transmission between these elements) should be secure. The system should be stable and resistant to virus and malignant code attacks.

Project organisation

Roles

Early on our group decided that the role of project manager would alternate every new phase and that everyone should be able to pick the role they most wanted. As no one seemed to be interested in the same role, distributing them in the group was simple. The final distribution can be found below.

Customer: RedRiver.
Tutor: Tobias Ohlsson.

Project manager: Alternating every new phase.
Inception project manager: Sofia Kristiansen.
Elaboration project manager: Linda Ott Olander.
Construction project manager: Andrew Galbraith.
Transition project manager: Jimmy Bengtsson.

Customer contact and requirements manager: Andrew Galbraith and Linda Ott Olander.
Technical manager: Jimmy Bengtsson.
Test manager: Sofia Kristiansen.

Development teams

In the beginning of the project Sofia Kristiansen and Linda Ott Olander were primarily responsible for the written documentation. Jimmy Bengtsson was responsible for the front-end of the application and Andrew Galbraith for the back-end. This was due to Jimmy and Andrew having the most experience with the chosen technologies i.e. ReactJS and ASP.NET. This worked fine, but in hindsight Sofia and Linda should have been more part of the

development and Jimmy and Andrew more part of the documentation from the inception phase in order to reduce the risks created by specialisation.

It was first at the end of the elaboration and beginning of the construction phase (iteration 4-5) that it was decided that Linda would work on the front-end with Jimmy and Sofia would work on the back-end with Andrew. Had we not put so much time and effort on the documentation and been stuck in the analysis paralysis for so long, we might have gotten further with the application.

Respective roles in the development teams became as shown below, with the person primarily responsible listed first.
Front-end: Jimmy Bengtsson, Linda Ott Olander, Sofia Kristiansen.
Back-end: Andrew Galbraith, Sofia Kristiansen.
Testing: Sofia Kristiansen, Linda Ott Olander.
Documentation: The whole group.

Implementation

Method

The method for the project has been a combination of Unified Process and SCRUM. The resources available were 4 people with 180 hours each to put into the project, a total of 720 hours.

The project was divided into four Unified Process-phases: Inception, Elaboration, Construction and Transition. Each phase was divided into weekly iterations. Inception had two iterations (not counting an initial start-up week), elaboration had three, construction had three and transition had two.

Meetings
The majority of communication was done through our own Slack-channel. This worked well and was necessary due to the fact that our group consisted of distance students.

We had weekly group meetings using Slack calls. Times for these meetings were decided the week before or the same day they were held in the group Slack-channel. Mostly the meetings were held on Mondays at 09:00 or 14:30.

We had weekly tutor meetings with Tobias Ohlsson using Slack calls. These were held every Tuesday at 09:00. They were then followed by a group meeting discussing what had been brought up during the tutor meeting.

Communication with the customer was done on a weekly/bi-weekly basis through emails and by weekly/bi-weekly Google Hangouts. These Google Hangouts were booked via email correspondence with the customer a week beforehand or during the end of a meeting. The information flow between

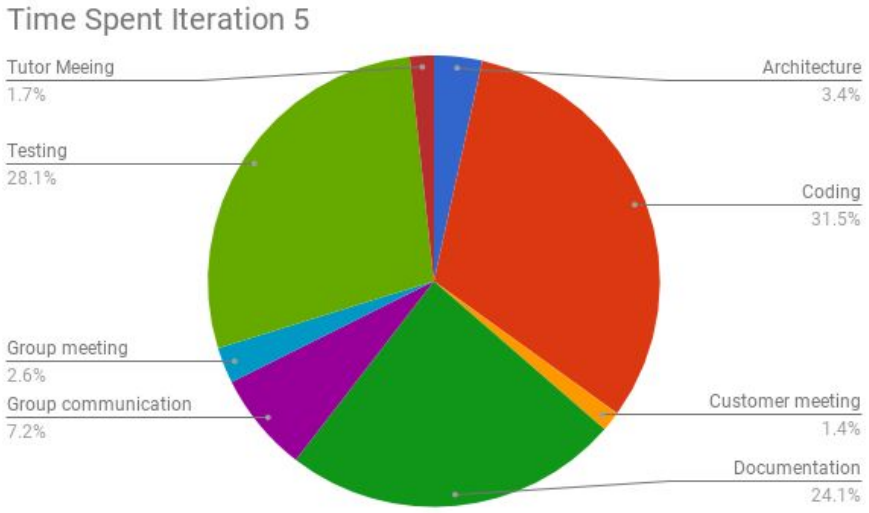
our group and the customer worked well and the group felt the amount of meetings to be adequate.

During every customer meeting we delivered the newest version of the system. Together with the customer we went through the application and asked for feedback concerning what was good and what should be worked on. Doing the deliveries like this worked well as we got the feedback instantly. When we instead sent a link to the live version for the customer to test on their own, the feedback was scarce and it was apparent that they did not have the time to sit down and test it on their own. Therefore the delivery during the customer meetings worked best for us.

All group, tutor and customer meetings have been very efficient and they have been held frequently. That said, each group member missed at least one meeting each and this could have been avoided if we had planned the time of the meeting better, taking into consideration all other plans that we might have.

Collaboration in the project worked well. Having weekly group meetings helped us structure that week's work. If anyone needed help the issue could be solved during that iteration and not be carried over to the next. Additionally, if anything came up after the meeting it could be discussed in the Slack-channel.

Time reports
During the inception and elaboration phase the individual time reports were written as tables in the Github wiki. In the construction and transition phase we started doing our time reports in a Google spreadsheet, to make it easier to calculate total time invested in the project and to make diagrams over what parts we mostly spent our time on. This made it a lot easier both to do our own time reports, but also to compile them into a group time report for each iteration.



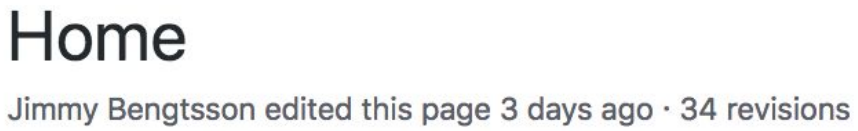
Changing from Github wiki to Google spreadsheets made it easy to make diagrams for how time was spent during the iteration.

Due to the size of the project it was hard to make accurate time plans. Looking at the individual time reports the estimates improved over the course of the project, at least when it came to recurring tasks like testing, meetings and implementation. When faced with a task that no one had experience of it was hard to make a good estimate.

Documentation
Project documentation was written in the Github repo wiki. Additional documents such as Google spreadsheets are linked in to the wiki so that they are easy to find. Google spreadsheets were mainly used for the product backlog so each group member could be part of prioritizing the requirements in the initial phases. This was done by letting each group member set their own priority number on the requirement, which was then calculated into an average. This average number was then used as the priority number. Allowing the whole group to individually prioritize the requirements did have the advantage that everyone became involved, and resulted in a good discussion on what requirements were most important for the application.

Google spreadsheets was also used to flag the requirements that needed further clarification by the customer; this was done after the group had commented on the requirements and after the test manager had checked if the requirements were testable or not. In the later phases we also used a Google spreadsheet for both our individual time reports and a compiled joint time report for the iteration, from which it was possible to extract diagrams showing how we had spent our time during the iteration.

One advantage of using the Github wiki is the resulting version control; it was possible to click on revisions to compare two versions of a document.



Github wiki version control

We received useful feedback on how to improve our documentation throughout the project both from a peer-review group and our tutor. Our group took the feedback to heart, changing aspects of the documentation that we felt were warranted, and we are pleased with the end results.

Testing
The overall goal of testing was to achieve 100 percent functionality coverage. This was to make sure that the system fulfilled the customer's requirements and worked as expected at the time of final delivery. Our goal was to at least carry out one test on every part of the system, whether it be automated, explorative, negative or other.

The aim was to carry out automated and manual tests on a weekly basis to catch bugs early. When a bug was detected they were tracked using the

Github repo issues so that everyone in the group knew what needed to be worked on.

The automated test suites and test cases were documented with the test code in our Github repo. These tests did not have any detailed test cases written for them since the test code was considered enough documentation. The manual test suites and test cases were documented in our Github wiki with detailed test cases. The test cases were named with the same number as the corresponding requirement, in order to create traceability between the product backlog and the test specification. As an example, test cases T.10.1 to T.10.6 were used to test requirement F10.

After the testing was carried out, a test report was written containing information like:

- date of the testing.
- a short description about what was going to be tested.
- name of the tester.
- version of the application.
- a description of the testing environment.
- a table like the one below.

Requirement	Test case	Status	Comments
F10	T.10.1	Pass/Fail	**Mac firefox, chrome, safari, iPhone 5s safari:** Pass. **PC edge, firefox:** Unable to make call to safari. **iPhone 5s chrome, firefox:** Unable to make call to safari, chrome, firefox.
F10	T.10.2	Pass/Fail	**Mac firefox, chrome, safari, iPhone 5s safari:** Pass. **PC edge, firefox:** Unable to receive or answer call from safari. **iPhone 5s chrome, firefox:** Unable to answer call from safari, firefox, chrome.
F10	T.10.3	Pass	**All devices:** Pass.
F10	T.10.4	Pass/Fail	**Mac firefox, chrome, safari, PC edge, firefox:** Pass. **iPhone 5s chrome, firefox:** Unable to test. **iPhone 5s safari:** Able to mute the microphone but not unmute it again.

- points of improvement.
- an analysis describing the feel of the system.

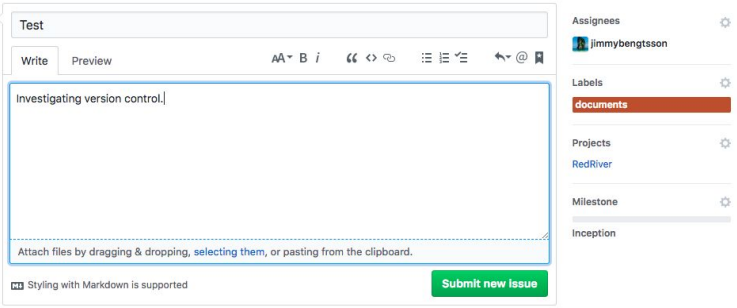
Version control

To be able to work on the same Github repo and source code we had to set up version control to avoid merge conflicts. Below is the version control in its entirety.

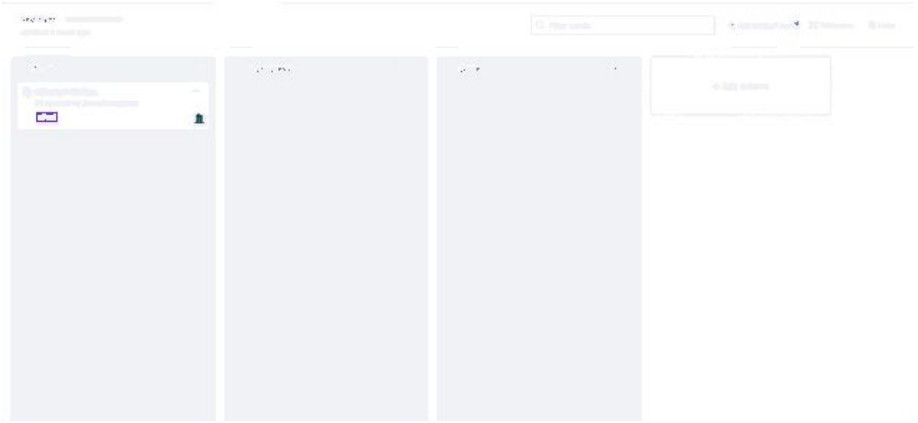
Note that some of the pictures are GIFs originally, but these don't work in the PDF-format. To see the version control with GIFs go here: https://github.com/jimmybengtsson/grupp03-redriver/blob/master/VersionControl.md

If you don't have the repository on your local machine:
$ git clone https://github.com/jimmybengtsson/grupp03-redriver.git

1. Create issue.
Create an issue based on the requirement to implement. You can also add assignees, labels, projects and milestones.



You can add the created issue to Project-page.



2. Create new branch

Before creating a new branch, pull the changes from upstream to your local master-branch. Your local master needs to be up to date.
$ git pull origin master

Create a branch on your local machine, named after the new issue, and switch to this branch :
$ git checkout -b [name_of_your_new_branch]

Push the branch to github :
$ git push origin [name_of_your_new_branch]

You can change working branch by:
$ git checkout [name_of_branch]

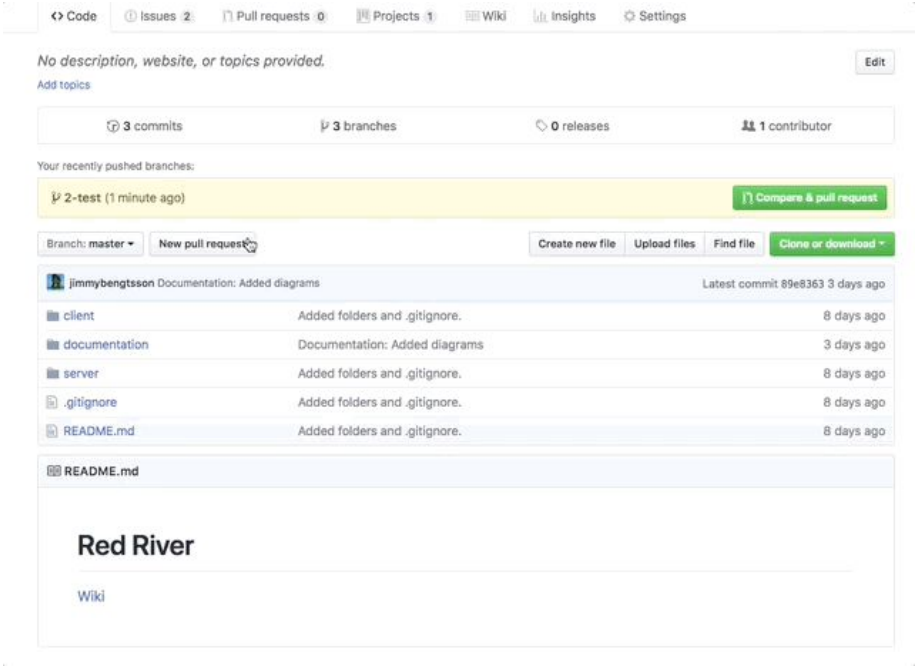
3. Commit and Push
Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a commit, and adding

them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

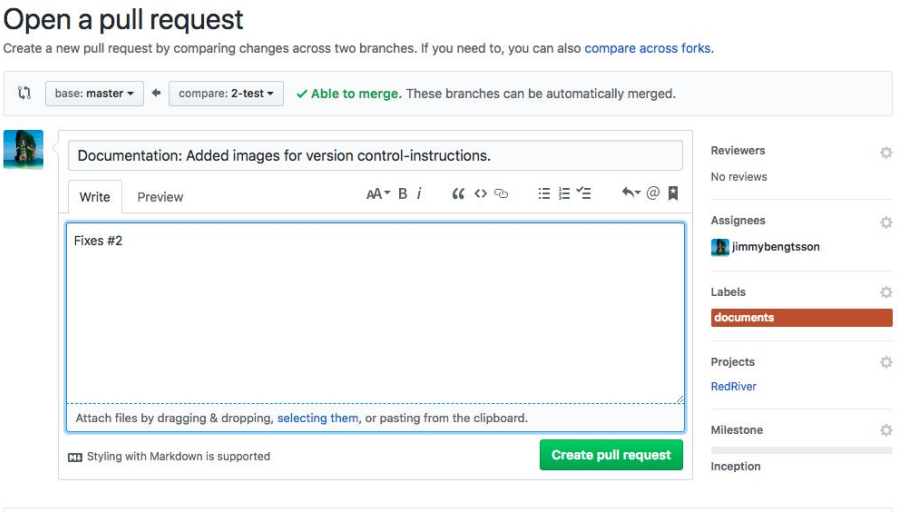
When you want to commit something, make sure to be in the correct branch. Not in master-branch! You can see all branches by using :
`$ git branch`

Add to Git and Commit as normal but use this command when pushing to Github:
`$ git push origin [name_of_your_branch]`

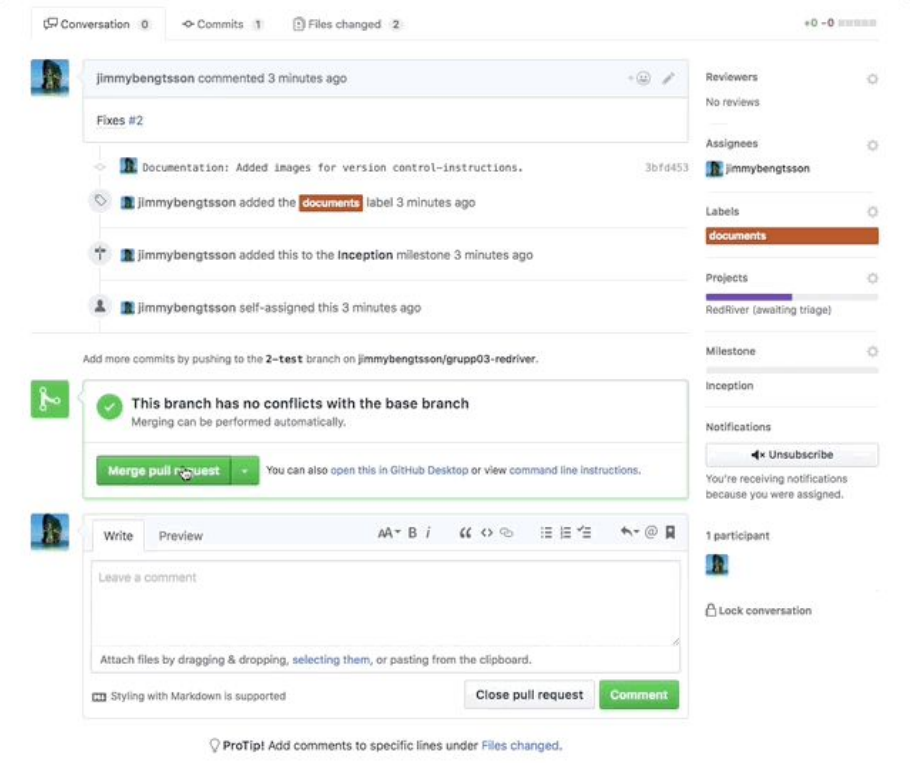
4. Pull request
When you're finished implementing your branch, create a pull request in repository on Github.



You can add the keyword `Fixes` and number of the issue to automatically close the issue after merge.



After the branch has been reviewed and everything seems ok, you can merge your created branch with the master branch.



5. Delete branch
If you want to delete the branch after merge, make sure to do it both on your local machine and on Github.

Local machine:
`$ git branch -D [name_of_branch]`

Github:
`$ git push origin :[name_of_branch]`

6. Read more about:
Github flow – https://guides.github.com/introduction/flow/
Create and delete branches – https://github.com/Kunena/Kunena-Forum/wiki/Create-a-new-branch-with-git-and-manage-branches
Pull request – https://help.github.com/articles/about-pull-requests/
Merge – https://help.github.com/articles/merging-a-pull-request/

Inception
During the inception phase we established contact with the customer RedRiver and discussed their requirements for the system they wanted us to build. The roles in the project were distributed. A Github repo was set up, a risk list, vision and project plan was written, preliminary software architecture diagrams were created, the wiki pages for the documentation were set up, the first product backlog was written and questions about the requirements were gathered in a document that was sent to the customer.

Version control for how to work as a group in the Github repo was created and technical solutions for the different parts of the system was identified.

Proof of concepts were developed and an api for the application was set up as well as some automated tests in Postman. The requirements were evaluated based on if they were testable or not and the first manual test cases were written. The requirements were prioritized in a Google spreadsheet by all four of the group members.

A prototype for the application was set up in Draw.io.

At the end of the inception phase the vision was signed off by the customer, large technical risks were mitigated and a presentation was held.

Elaboration
During the elaboration phase, after both holding our own presentation and seeing the other groups' presentations concerning the kind of systems they were to build, we understood that the project was too big for us to be able to finish within the limited project time. During customer meetings we asked them to limit their requirements and the requirements in the Google spreadsheet were re-prioritized according to which requirement that needed to be implemented first. Manual test cases for the highest prioritized requirements were created and automated Jest tests for the client were written and carried out.

The application's API was further developed and tested.

A proof of concept for the video calls was created and tested in different browsers and operating systems.

The prototypes set up during the inception phase were finished and presented to the customer during a meeting, where they gave us positive feedback stating that we were on the right track.

The client, server and database were set up on Andrew's own Azure account, in order for us to have a live version of the application.

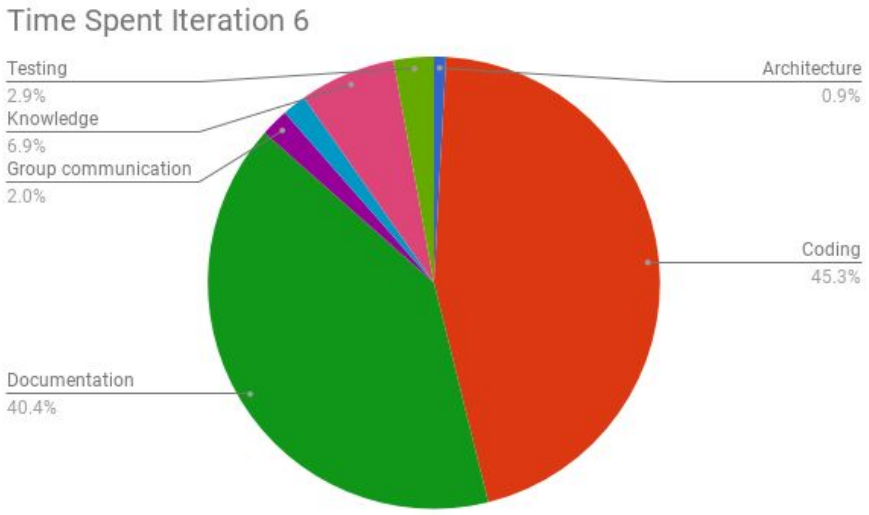
Functionality in the form of registering and login was implemented on the client and the decision to use Material-UI for the design and layout was made.

The rest of the phase was meant for us to decide how to technically implement the customer's requirements. The software architecture was confirmed and delivered. Routines for testing were up and the project was estimated to be feasible based on the software architecture.

At the end of the elaboration phase we had implemented requirements F1-F4, which was composed of registration, login, seeing one's own user information and adding friends.

Construction
During the construction phase another presentation was held with a focus on the technical aspects of the system.

Iteration 6

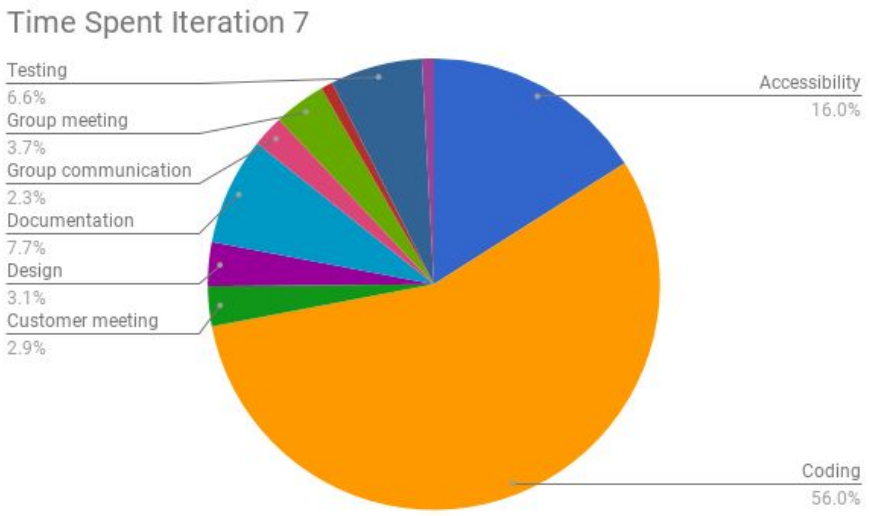


The goal during the first iteration in the construction was to implement chat functionality. However, we had some problems concerning SignalR and its use in the project, which was due to the use of differing versions of SignalR on the client and server. This cost us a lot of time, since troubleshooting was difficult. The requirements concerning chat functionality were therefore not implemented in the first construction iteration, although the code was done on the back-end and the groundwork was laid out on the front-end.

The first iteration of the construction also saw Sofia and Linda become more involved in the coding, going through the code to get a overview of how the code was structured.

During the iteration requirement F47, the ability to upload a profile picture, was implemented on the back-end and front-end. Further tests were created and carried out on existing functionality and all bugs were documented in the Github repo as issues to be worked on.

Iteration 7



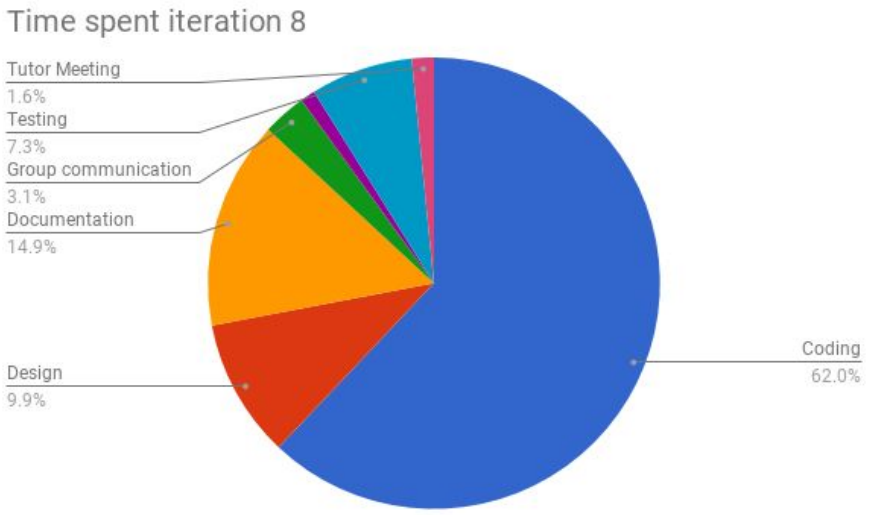
During the second iteration of construction the focus was on delivering the system to the customer, as we were approaching the last week of construction and felt that it was important to communicate to the customer how far we had come in terms of implemented requirements; it was also important to reach an agreement about the state of our "final" product (in that we have only had time to implement a fraction of the total requirements) and the nature of the product final delivery.

The goal of the demonstration was to show working chat functionality and so a lot of time was spent making sure the product was in good shape ahead of the delivery demonstration. Jimmy and Andrew worked on the client and server-side respectively to eliminate various issues that had cropped up during the past weeks. Sofia continued her testing duties but also wrote server-side code to allow avatars to be uploaded. Andrew and Linda began to look at the WCAG 2.1 guidelines in order to understand what needs to be done in order to conform fully.

Linda also created mock-ups of the client with the aim of improving its appearance, both from an aesthetic and accessibility viewpoint.

The delivery went well, and we felt that we communicated clearly to the customer our thoughts around the status of the project and its future development.

Iteration 8



Focus during the last construction iteration was to correct bugs that had been discovered during testing and to update the design according to WCAG. We managed to remove many of the bugs reported in issues and also implemented some features that had been left out earlier in the project.

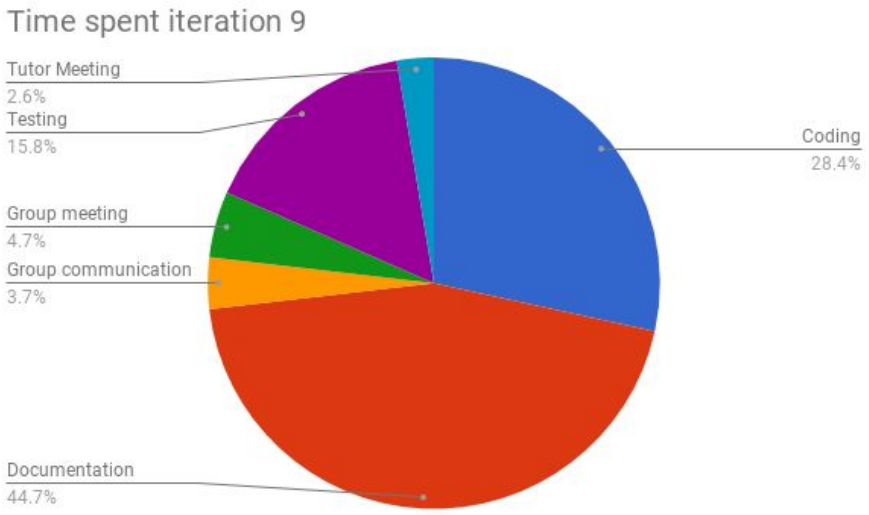
Andrew added API-routes to the server for changing a users details and password while Jimmy added these features to the client. Sofia worked on requirement F13, "Consent email", so that an account verification email was sent to the user after registration. Linda started working on requirement F20, "User account deletion", and updated the design throughout the application according to WCAG. Text, buttons and colors were updated.

Both Sofia and Andrew added tests for newly implemented requirements. Sofia created manual test cases for some of them and Andrew continued writing Postman-tests for the server.

Some documents were also updated before handover to group 0 for peer review. Updates were made to the technical documentation with new diagrams to match changes in the code and instructions for installing, deploying and hosting both server and client was created.

In the end of the construction phase the chat, the ability to upload a profile picture, a consent email being sent out after registering and video calls was in place. The implementation concerning user account deletion was well under way and a lot of the bugs found during testing was solved.

Transition
In the first iteration of the transition phase the majority of group members had to focus on another course and this slowed development. The time we put in was mostly spent on documentation and updating the wiki to prepare for the final handover and delivery to the customer.



Our goal for the week was to test as many of the requirements as possible from the ones implemented during the last iteration, trying to solve bugs detected during the testing. Sofia carried out the manual test cases for requirements concerning chat functionality and profile picture upload and then wrote test reports for them. Linda did some manual testing on Android mobile and iPad and wrote a test report. Issues were created on Github for the failed tests and a number of these were solved. Andrew started to load test the server by using the testing tool JMeter. He also wrote tests for SignalR which differs slightly from earlier testing on the server because SignalR uses WebSockets instead of HTTP to communicate.

The customer carried out a final acceptance test and accepted the system.

During the last iteration of the transition phase our end presentation was held; the final report was also written. The server was stress-tested, regression testing was carried out on all of the testable, implemented requirements and documentation for the project on the Github wiki was finished. Minor bug fixes were carried out. The iteration ended with a customer meeting for the final delivery.

Technology

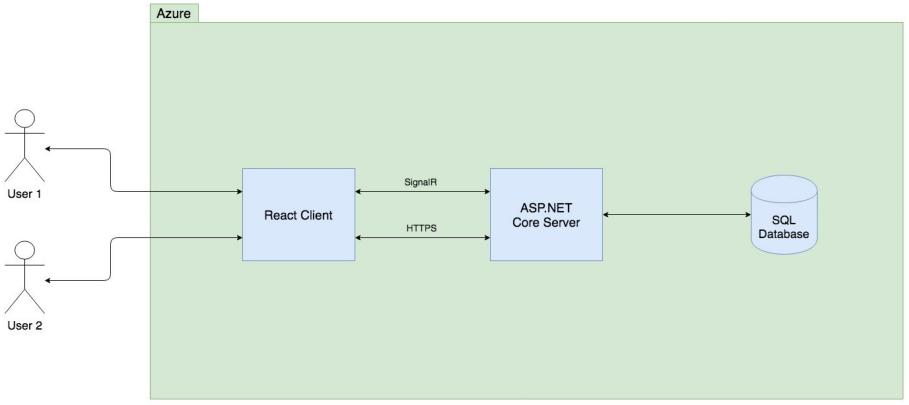
Client
The client application was built on the React.js-framework with React Router for routing. When using React, handling navigation gets really simple with the help of state-handling and React Router. In React, it is also easy to create components that can be reused, which gives a consistent look and feel throughout the application and follows the DRY-principle. React also includes some great developer tools that can be installed as Chrome-extensions to facilitate the development.

Server
The RedRiver server was primarily designed as an api server, and so the client and server communicate through API calls. However, the messaging

system used SignalR. The base server implementation used ASP.NET Core 2. This had the advantage of being compatible with the majority of operating systems, in contrast to earlier ASP.NET variants.

Database
The RedRiver server used a Microsoft SQL relational database for all data storage. This had the advantage of playing nicely together with ASP.NET and was the default choice in this context.

Hosting and architecture
The client, server and database used in the project were all hosted within Microsoft Azure according to the following diagram.



Result

In our opinion the project has gone well, without any major setbacks or significant technical problems. Any technical problems which did arise, for example CORS problems within the hosting environment or differing SignalR versions, were dealt with in a timely manner and did not cause undue delay to the project.

However, the scale of the project was sizeable from the outset, and we have been candid with the customer as to what we believed was possible within the project's timeframe. In this regard, our predictions were accurate and we have implemented only a part of the total required functionality.

We delivered a working client-server-database architecture where the user can:

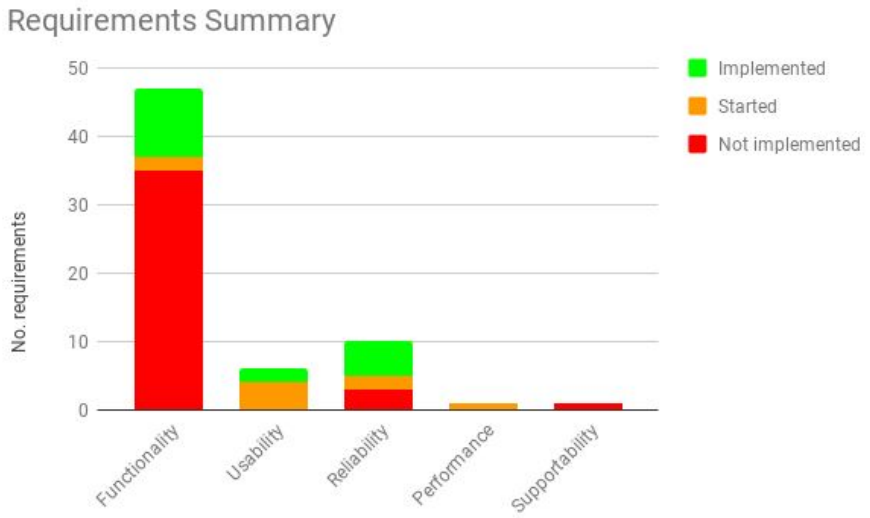
- register (including email verification), login/logout, update details (including avatar), change password.
- add/delete friends.
- create, join and leave chat groups.
- send private (HTTPS) chat messages with chat groups, where no chat messages are stored on the client device.

We also delivered limited video chat functionality that has been tested and where the known bugs has been documented for the customer.

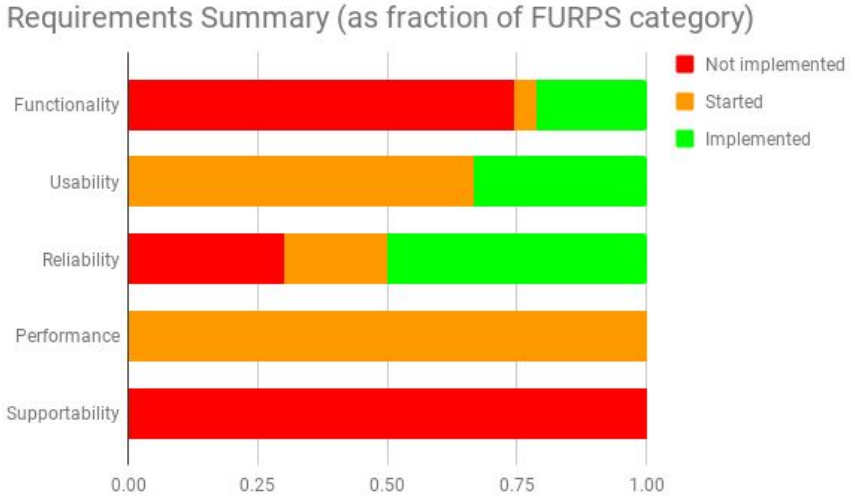
The implemented requirements are summarized in the following table:

Status	F	U	R	P	S	Total
Not implemented	35	0	3	0	1	39
Started	2	4	2	1	0	9
Implemented	10	2	5	0	0	17
Sum	47	6	10	1	1	65

Table showing the amount of requirements based on the type and their status at the end of the project.

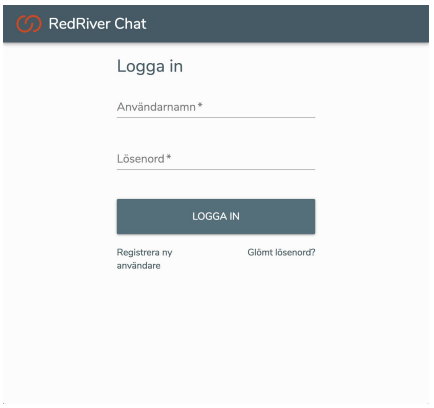


Summary of total number of requirements implemented per FURPS category.

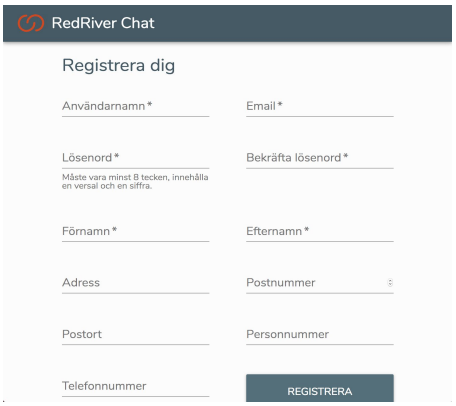


Summary of number of requirements implemented as fraction of FURPS category.

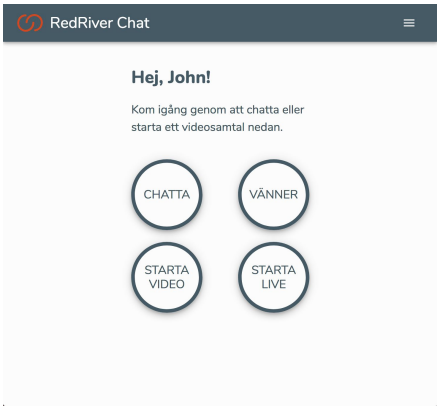
Screenshots of the delivered system



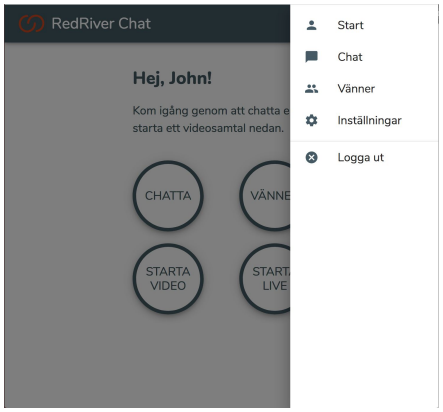
Login page/home page



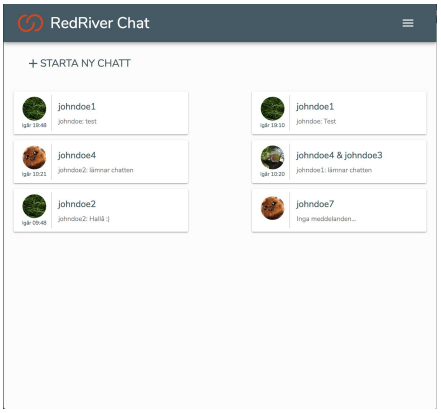
Register page



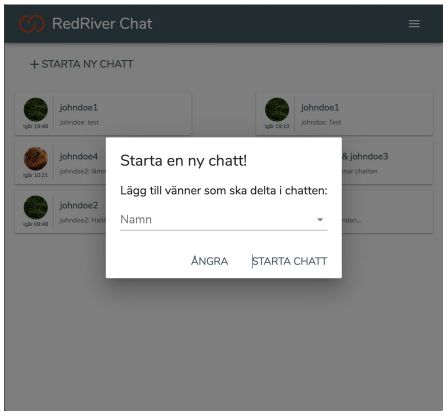
Logged in user/home page



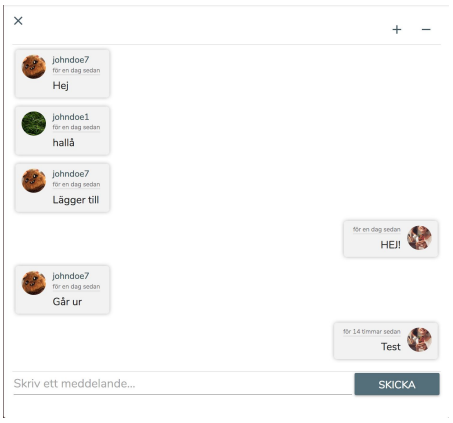
Side menu overlapping user page



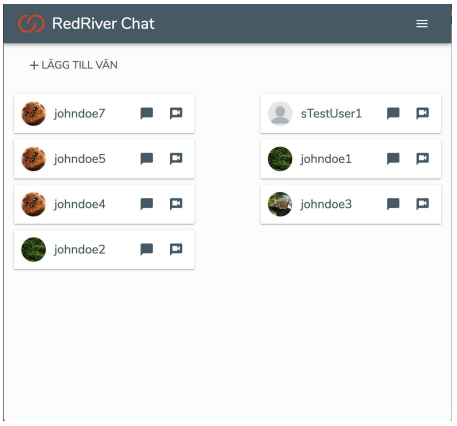
Chat page with active chat rooms



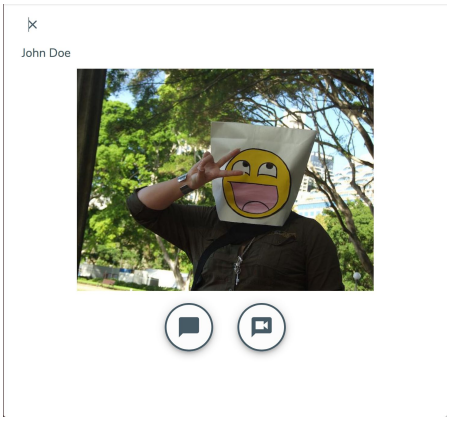
Interface for creating a new chat room



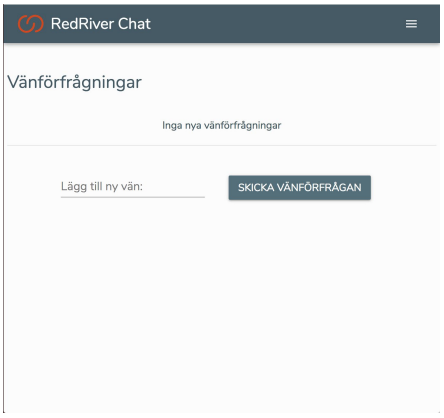
Chat room with multiple users



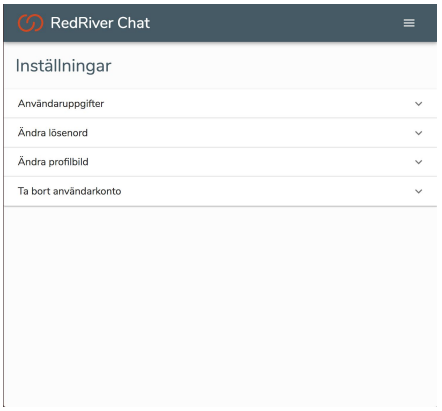
Friends page



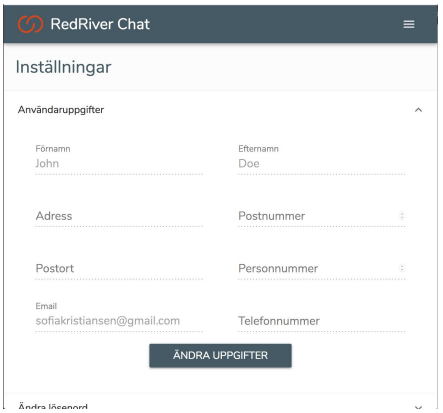
Friend details



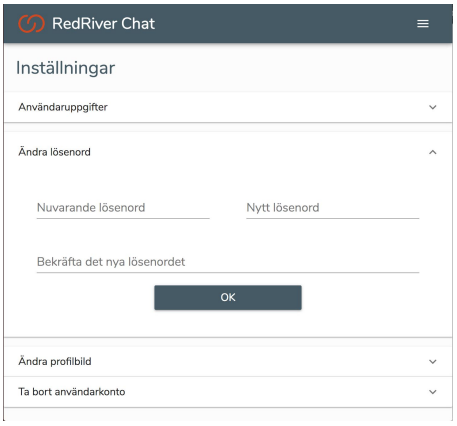
Friend request page



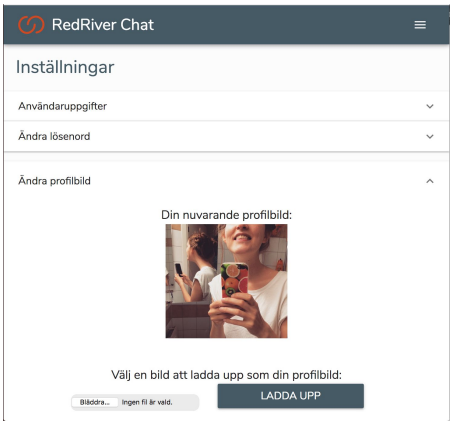
Settings page with dropdown list



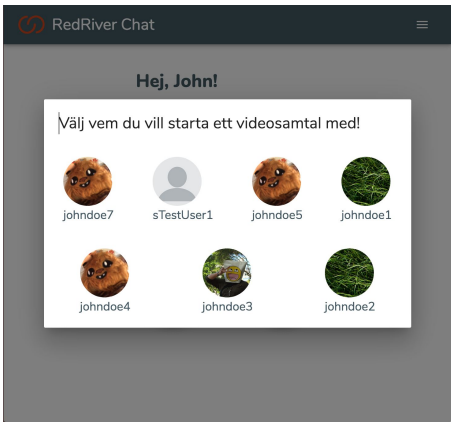
Settings page - User information



Settings page - Change password

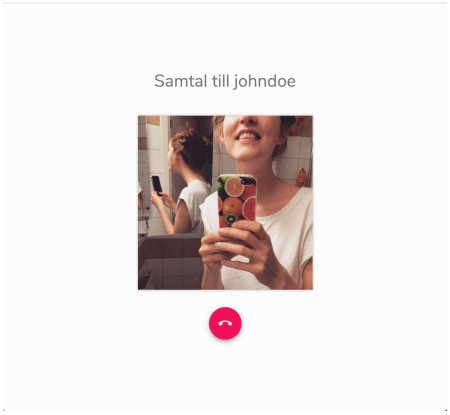


Settings page - Upload profile picture



Interface for choosing who to start a video call with

Video call - Receiving call from another user



Video call - Making call to another user

Deviations

Upon receiving the product description from the customer it was apparent that the scale of the project would be greater than the allocated time; upon entering the last phase of the project it was confirmed that this was in fact the case. In terms of requirements we implemented only 26% of the total product backlog. This may not be regarded as a deviation in itself since it was predicted from the outset by our group and the customer was made aware of this eventuality early on in the process.

However, one consequence of such a large project, and its associated sizeable requirement list, was a need for reliable documentation to ensure a common frame of reference for our group and our customer. This documentation required a large proportion of iteration time early on in the process, but even later iterations saw a significant amount of hours devoted to writing documents (40% of time spent in iteration 6, for example).

Another consequence of the size of the project was the need for communication within the group, especially during early project phases, in order to decide upon roles, task assignments and so forth. Being a distance group and communicating primarily by internet chat, such communication perhaps took a longer time that would have been the case for campus based groups. Although chat based communication, and Slack in particular, certainly had its advantages, one disadvantage was the time taken for even simple decisions to be agreed upon.

In both the case of documentation and group communication the time spent was more than our group had originally reckoned with, although in hindsight this was very much a tradeoff. Documentation and communication were necessary, above all early in the process, to establish project structure and working routines, and had we reduced the time spent on these aspects by too great a degree it may have had an adverse effect on the product.

Conclusion

The scale of the project was sizeable from the outset, which perhaps led to “analysis paralysis” where too much time was spent analysing requirements; however, having talked to the customer about the size of the product backlog it was then possible to begin coding in earnest.

Early on we researched what technology or tools were required for implementation of key project areas e.g. chat functionality, video calls, encryption etc. By creating proof of concepts we felt comfortable with the technology we had chosen. This was also a good way to mitigate the potential larger technical risks.

By initiating testing as early as we did instead of waiting until everything was implemented, our confidence in the application and the implemented parts grew.

Even though it has been hard sometimes to keep going we are pleased with our efforts and that, in spite of all the difficulties, we were able to create a functioning application that the customer is happy with.

The project has been challenging but it has also been a great experience to work in a group on a real life project; we have learned a lot from each other and it has been highly educational to experience working with a customer that might not have a clear idea of what they really want.

In our opinion the project has gone well, without any major setbacks or significant technical problems.

Suggestions on further development

We believe that both our code and documentation set a solid ground for further development, both in terms of the available source code and documentation. Going forward, we recommend the following:

SignalR at present uses an in-memory dictionary to store current connection details. This is fast and suitable for lower volumes of users. Should the number of users increase greatly then this might not to be the best solution; instead the architecture must be scaled out. Documentation concerning this is available here: https://docs.microsoft.com/en-us/aspnet/signalr/overview/performance/scale-out-in-signalr.

In our project a Microsoft SQL database was used for all purposes. The pros and cons of this should be evaluated against document databases such as MongoDB. Should the decision be made to continue with a relational database, the tables used should be normalised. For example, a user can enter into a friendship with another user; at present this is stored as two entries in a friendship table (one friendship in each direction). Alternatively this might be seen as a single database entry between two users, thereby offering substantial space savings in a system with many users.

Encryption: Our project featured communication over HTTPS and built in database encryption, and no data is stored on the client device. However, database admins can still see chat logs. End-to-end encryption would solve this problem. Our initial research showed that this would be difficult to implement in a browser based system, but some form of partially encrypted logs is fully possible.

Video chat is available, although it is not fully tested and may not work on all devices. Going forward, this should be further investigated and well-tested so as not to lead to an adverse user experience.

SendGrid is used for sending verification emails to a newly registered user. For it to work properly after the handover it needs to be set up. How to do that is described in the wiki documentation named “SendGrid”: https://github.com/jimmybengtsson/grupp03-redriver/wiki/9.8-SendGrid.

Fixing the known bugs documented as issues in the Github repo:
https://github.com/jimmybengtsson/grupp03-redriver/issues

Organisation taking over

At the end of the project the application will be taken over by the customer RedRiver. We have expressed to the customer that we are willing to answer a certain amount of support questions concerning functionality, implementation etc. should they wish to further develop the product.

At the handover the customer will get:

- A copy of the source code.
- A downloaded version of the documentation wiki.

The customer also has, and will continue to have, access to the Github repo where both the source code and wiki are maintained.

Literature suggestions/documentation reference

Documentation concerning the project can be found in the Github repo wiki:
https://github.com/jimmybengtsson/grupp03-redriver/wiki

Installation client:
https://github.com/jimmybengtsson/grupp03-redriver/blob/master/client/README.md

Installation server:
https://github.com/jimmybengtsson/grupp03-redriver/blob/master/server/README.md

Documentation technology and tools:
Links to the technology and tools can also be found in the Github wiki.

- ASP.NET core 2.1
  https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1
- ASP.NET HTTP:
  https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http?view=aspnetcore-2.0
- Axios: https://github.com/axios/axios
- Azure: https://azure.microsoft.com/sv-se/
- Azure portal: https://portal.azure.com
- Azure Cosmos DB:
  https://docs.microsoft.com/sv-se/azure/cosmos-db/introduction
- JWT: https://jwt.io/introduction/
- Material-UI: https://material-ui.com/
- ReactJS: https://reactjs.org/
- React Routing: https://github.com/ReactTraining/react-router
- SendGrid: https://sendgrid.com/
- Signal Protocol library:
  https://github.com/signalapp/libsignal-protocol-javascript
- WebRTC: https://webrtc.org/start/

- Websockets (SignalR Javascript Client): https://docs.microsoft.com/en-us/aspnet/signalr/overview/guide-to-the-api/hubs-api-guide-javascript-client
- Websockets SignalR: https://www.asp.net/signalr

The application API:

- RedRiver API (written in the project): https://documenter.getpostman.com/view/1600195/redriverserver-api-documentation-v21/RW86L9vn

Improvement suggestions for future projects

Decide early on who works with what in the project to get a better distribution of tasks in the group. Our problem was that we didn't involve all group members in the code development before iteration 5, as well as some members taking more responsibility for the documentation.

Using tables in the wiki is good, but very hard to read and update when shown in .md-format. Google Sheets was a better alternative for our project and group.

Don't get stuck in analysis paralysis. Even though the requirements might be unclear at first, start to implement the basics like registering and login early.

In the beginning of a project, before the group has had a chance to meet with the customer, there is often some extra downtime. A good way to use that time is to figure out which tools and methods will be used in project management. One thing that saved us a lot of time was to move the time reports from our wiki to Google Sheets. During the peer review process we also had the opportunity to view how the other groups worked with their project management, what tools and methods they used. Putting some extra effort into thinking this through in the beginning of the project, how to use methods and tools in the best way, can save a lot of time which can be spent on coding and other parts of the project.

We've been very pleased with our use of Github and issues. The issues have been used as a task manager for the requirements to keep track of what needs to be implemented and to see what the others in the group was working on at the moment. It has also been used to keep track of bugs found during testing, which gave us an overview of what needs to be fixed and who is fixing it.

On Github we also used the kanban boards found under the "Projects" tab, which made it easier to see what issues were being worked on at the moment, what needs to be done and what was already done.

Don't be afraid to try different workflows. When changing project managers each phase we all got to decide how to work on the project. Trying these different styles of workflow has been instructive and it opened our eyes to the use of other tools and other ways of working.

Decide when and what to deliver to the customer so that you have clear deadlines to structure the work around.

Appendix

- RedRiver Chat Demo: https://youtu.be/UNYPgCy1XlI