SA Documentation

Monad Torrent

Semester: Autumn 2024

Version: 01.00
Date: 2024-12-20 16:06:38Z
Git Version: 9da5147

Project Team:	Davor Lucic
	Fadil Smajilbasic
Project Advisor:	Dr. Farhad Mehta

School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Contents

Part I

Management Summary

Management Summary

Even though this is the first chapter of your document, it is typically the last one filled with content. The *Management Summary* is a **brief and high-level summary** of your project. It should give any reader unfamiliar to the project an overview of the contents included later in the document.

A common structure is:

- What is the problem we wanted to solve?
- How did we solve the problem?
- Does your solution solve the problem in a successful way?
- Will there be consecutive projects based on your work?

Diagrams and images work very well in this chapter, especially screenshots of your software.

One final remark: a well written management summary is a good starting point for your **Project Presentation**, as you will address a similar audience there.

The Problem

From a product standpoint it is nothing new. There are many Bittorrent clients out there. From that standpoint it is of less importance in what language it is written and what libraries are used. The end user just wants a product that works.

When looking at it from a project perspective the problem looks very promising. Many students shy away from using the functional programming paradigm, as many view it as too difficult, too tedious, or they just simply don't see the point in switching from what they know already. We want to give our fellow students a different perspective, and maybe some courage to start their first functional project themselves.

The Solution

To better understand the intricacies of a functional project, we had to build one ourselves first. Because neither of us has any experience in building a larger product in Haskell, it gives us the unique advantage that we don't just understand the problem, but also feel it. By building the Monad Torrent we gathered the necessary experience to at least lay the groundworks of how a product built in a functional manner could look like. In addition to this documentation, we will also write a manual on how to recreate the glorious Monad Torrent, batteries included.

The Result

Even though we didn't cover every function we initially planned to implement, we believe this project was still a success, since we believe that a solid documentation is just as important as a working product. We hope that anyone that is interested in pursuing this further can use our manual as a foundation to extend their own Monad Torrent. Our goal during the creation of the product and manual was to approach each problem with the same approach, knowing we built a solid foundation to do so.

The Future of Monad Torrent

Upon submission we unfortunately have to put this project on ice for now. Next semester both of is will not have time to pursue this topic further, as we have other classes and projects coming up. Nevertheless, we don't want to flag it as "done" or "abandoned" quite yet, since there is still a lot of room for improvement, like magnet link functionality, multi-file torrents, UDP-friendly trackers, prettier UI, and so on.

Part II

Product Documentation

Chapter 1

Requirements

1.1 Actors

Unlike in many other application with different use cases for different roles, we want to focus on only one actor: the end user. Even in a team setting where many users would use this product, each and every user manages their files and usage of Monad Torrent independently.

Actor	Explanation
End User	• Can add torrents – Via link – Via file – Choose wich files to download • Can manage torrent queue – Reorder torrents – Pause/resume downloads – Delete torrents – Access torrent information (Size, Seeders/Leetchers, Torrent Contents) • Can access and change torrent settings – Via Settings menu – Via config file

1.2 Functional Requirements

FR-1	Downloading a Torrent via file
Actor	End User
Goal	Successfully download a torrent by uploading a torrent file
Preconditions	• The user has a valid .torrent file available
Postconditions	• The torrent file begins downloading to the user's device • The user can view the download progress and access the downloaded file once completed
Priority	High

FR-2	Downloading a Torrent via magnet link
Actor	End User
Goal	Successfully download a torrent by using a magnet link
Preconditions	• The user has a working internet connection • The user has a valid magnet link available
Postconditions	• The torrent file begins downloading to the user's device • The user can view the download progress and access the downloaded file once completed
Priority	Low

FR-3	Reorder torrents in download queue
Actor	End User
Goal	Change the priority/order of torrents in the download/upload queue
Preconditions	• The user has one or more torrents added to the queue • The user is using the Monad Torrent client with multiple active or pending torrents
Postconditions	• The torrents are reordered based on the user's preferences • The new queue order is reflected in the torrent client
Priority	Low

FR-4	Pause and resume torrent downloads
Actor	End User
Goal	Pause an active torrent download and resume it later
Preconditions	• The user has one or more torrents actively downloading or uploading • The torrent client is running
Postconditions	• Paused downloads stop consuming bandwidth, and downloads can be resumed from the last point
Priority	Low

FR-5	Delete torrents from the queue
Actor	End User
Goal	Remove a torrent from the download or upload queue
Preconditions	• The user has one or more torrents in the queue
Postconditions	• The torrent is removed from the queue, and optionally, its data is deleted from the disk
Priority	Medium

FR-6	Access General Torrent Information
Actor	End User
Goal	View detailed information about a torrent, including file size, number of seeders/leechers, and the contents of the torrent
Preconditions	• The user has added one or more torrents to the client • The torrent is actively downloading, uploading, or is completed but still in the queue
Postconditions	• The user views the requested information, which helps them understand the progress and health of the torrent
Priority	High

FR-7	Delete Torrent Files and Their Contents
Actor	End User
Goal	Permanently remove torrent files from the file system, along with any downloaded content (e.g., media files, documents, etc.)
Preconditions	• The user has already downloaded files using a torrent client • The torrent client can access the download folder • The user has sufficient permission to delete files from the file system
Postconditions	• The user views the requested information, which helps them understand the progress and health of the torrent • The torrent file is removed from the client and the system (if applicable) • The downloaded files and folder associated with the torrent are deleted from the file system • Disk space is freed up after deletion
Priority	High

FR-8	Adjust Download Speed via Settings menu
Actor	End User
Goal	Modify the maximum download and/or upload speed limits for torrents
Preconditions	• The torrent client has the capability to set speed limits • The user has access to the settings menu
Postconditions	• The download/upload speeds are adjusted based on the user's input
Priority	Low

FR-10	Adjust Download Speed via config file
Actor	End User
Goal	Modify the maximum download and/or upload speed limits for torrents
Preconditions	• The torrent client has the capability to set speed limits • The user has access to the config file
Postconditions	• The download/upload speeds are adjusted based on the user's input
Priority	Low

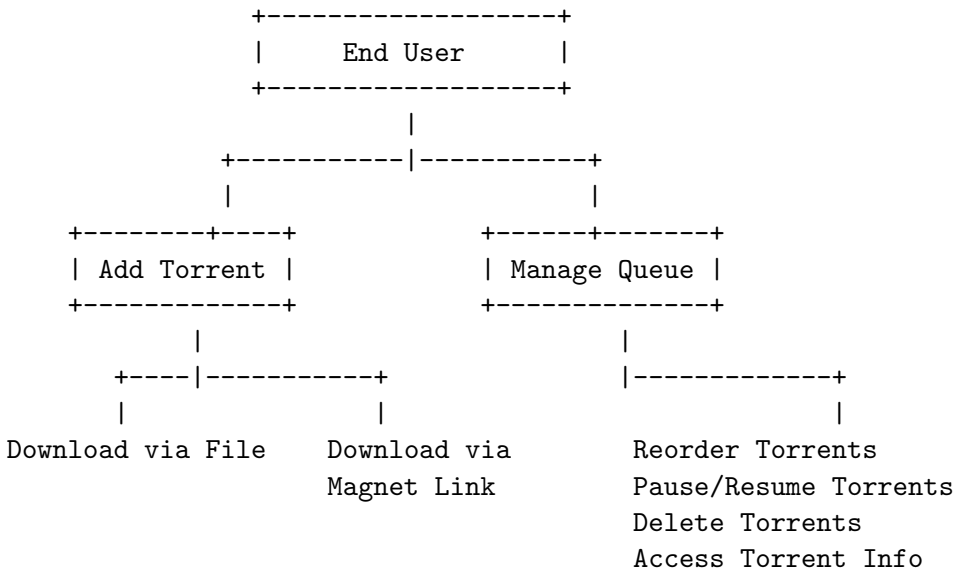
FR-11	Adjust Default Folder via Settings menu
Actor	End User
Goal	Change the default folder where torrents and their downloaded files are saved
Preconditions	• The torrent client allows the user to modify folder paths • The user has access to the settings menu
Postconditions	• The new folder location is used for all future downloads
Priority	Low

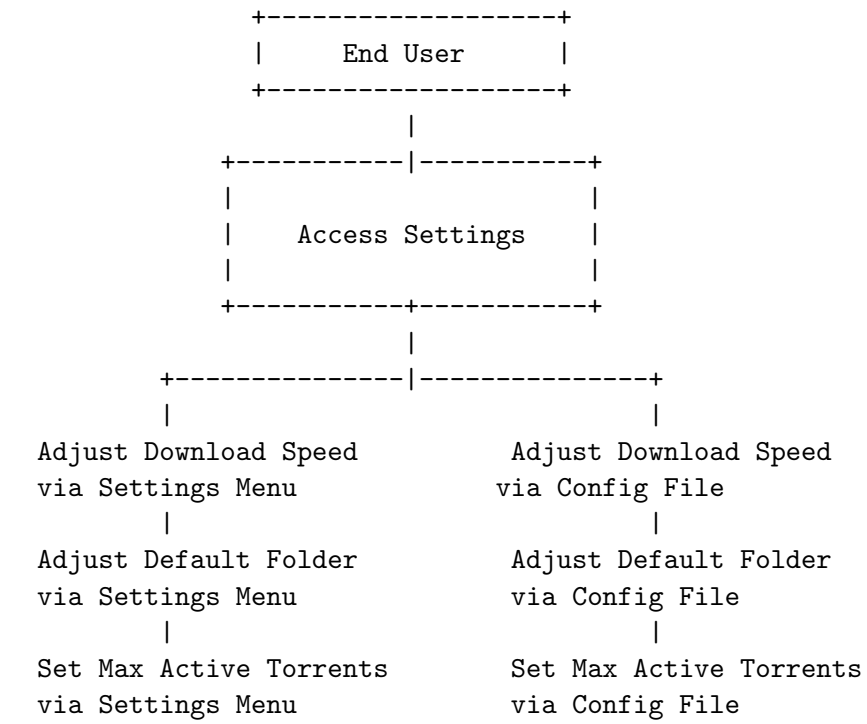
FR-12	Adjust Default Folder via config file
Actor	End User
Goal	Change the default folder where torrents and their downloaded files are saved
Preconditions	• The torrent client allows the user to modify folder paths • The user has access to the config file
Postconditions	• The new folder location is used for all future downloads
Priority	Medium

FR-13	Adjust Maximum Active Downloads via Settings menu
Actor	End User
Goal	Set the maximum number of torrents that can be actively downloading at the same time
Preconditions	• The torrent client allows the user to control the number of active downloads • The user has access to the settings menu
Postconditions	• The system enforces the maximum limit for active downloads based on the user's input
Priority	Low

FR-14	Adjust Maximum Active Downloads via config file
Actor	End User
Goal	Set the maximum number of torrents that can be actively downloading at the same time
Preconditions	• The torrent client allows the user to control the number of active downloads • The user has access to the config file
Postconditions	• The system enforces the maximum limit for active downloads based on the user's input
Priority	Low

1.3 Overview Use Cases





1.4 Use Case Diagram

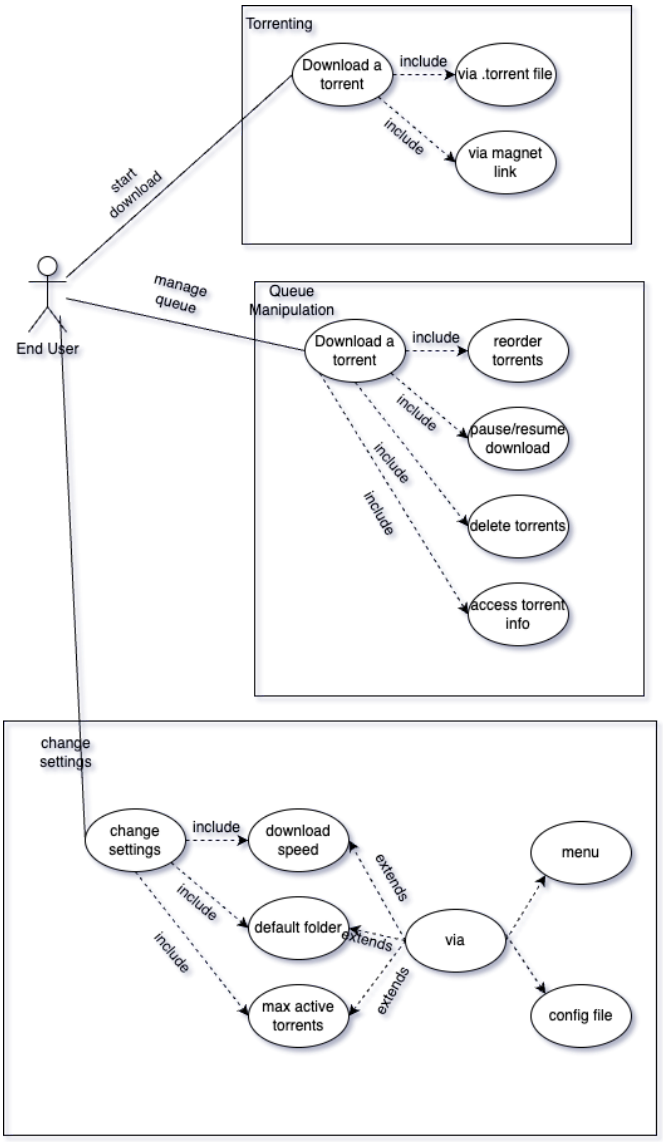


Figure 1.1: Use Case Diagram

1.5 Non-Functional Requirements

ID	NFR-1
Subject	Good target response time
Requirement	Performance - Time Behavior
Measures	• Response time propagating to the persistence storage below 500ms – defining limited scopes for default searches
Priority	Medium
Verification	• Extend testing to check response times meet target range

ID	NFR-2
Subject	Easy and intuitive usability
Requirement	Usability - Operability User Interface Aesthetics
Measures	• Layout should be intuitive and well labeled • Instant gratification with feedback information • User should be able to ”zoom” through the procedure without extensive training
Priority	Low to Medium
Verification	• External 3rd parties will be asked for feedback and get back min. 70% of positive Feedback • Feedback will be gathered by a form where Design, Usability and Customization will be rated from 1 to 10 • Another critical feedback question will also be ”Would you use this torrent solution over more-established programs?” or ”Would you recommend this torrent solution to others”

ID	NFR-3
Subject	Decoupled code
Requirement	Maintainability - Modifiability
Measures	• As little as possible coupled code, no big dependencies or global variables
Priority	High
Verification	• Thorough unit and regression testing • Specifically removing functions to check if they are unintentionally coupled

ID	NFR-4
Subject	Readable code
Requirement	Maintainability - Modifiability
Measures	• Interchangeable methods/classes even if metadata is parsed. Metadata should be an external object.
Priority	High
Verification	• With a linter check if amount of rows in functions are below 30

ID	NFR-5
Subject	Testable code
Requirement	Maintainability - Testability
Measures	• Testing of all boundaries (unit testing) • Regression testing for coupling references • Integration testing for ”final product” • Testability provided with CI/CD
Priority	High
Verification	• Every return value needs a unit test - boundary and type test • Input and Output validations - part of defensive programming • automated testing - CI/CD

1.6 Minimal Viable Product

As a first step into the development of this project we want to create a minimal viable product (MVP) that we can use to download a file. The MVP includes the following features:

- Parsing a torrent file
- Contact with tracker
- File piece request
- File assembly

Chapter 2

Domain Analysis

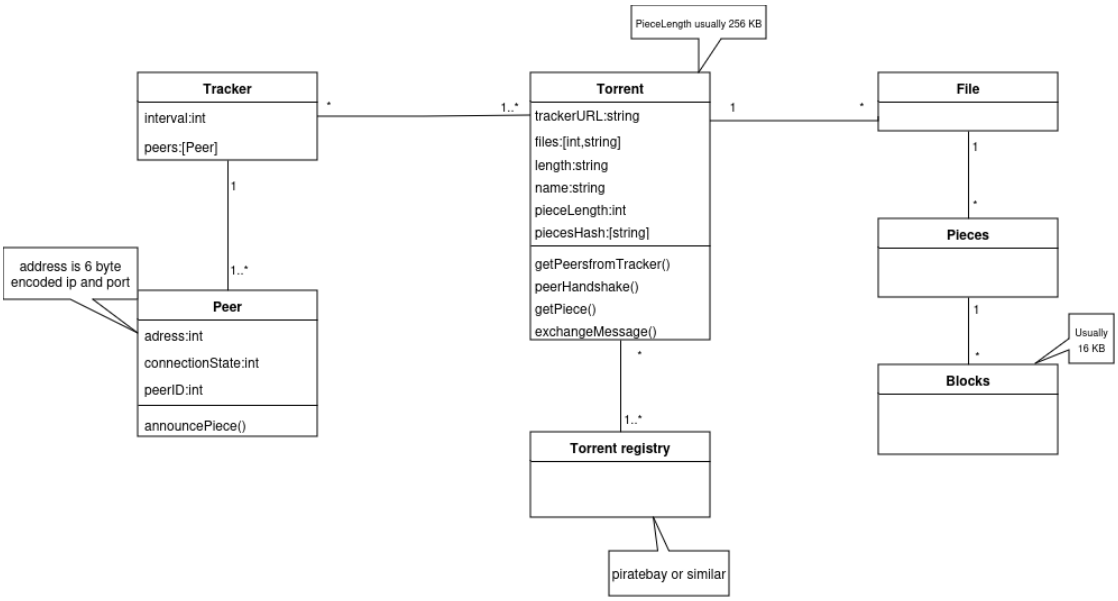


Figure 2.1: Domain Model Diagram

The domain model describes the torrent network and how a client can access and download a file. A file is made out of pieces which are made up of blocks of typically 16 KB in size. Firstly the client Accesses a Torrent repository from which he receives the Torrent file containing all the torrent information, like the tracker URL, the file name, the file size, the piece size, and the pieces hashes. Afterwards the client contacts the tracker and receives a list of peers that are currently seeding the file and are available to download from. The client then connects to the peers and starts downloading the file. The client downloads the pieces from the peers and verifies the hash of the piece. If the hash is correct the piece is written to the file, if not the client requests the piece from another peer. This process is repeated until the client has downloaded the whole file.

The client can also seed the file to other peers after he has downloaded the file.

Chapter 3

Architecture

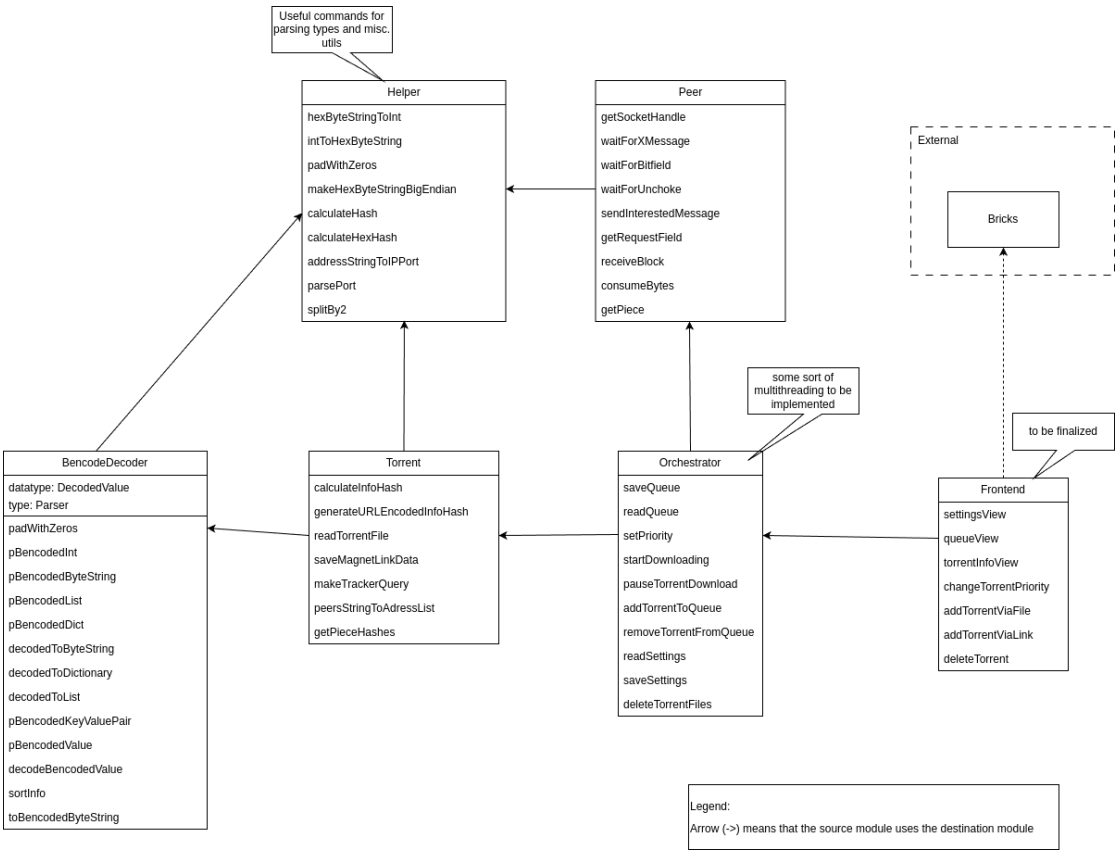


Figure 3.1: Architecture Diagram

We heavily leaned on the codecrafters.io guide as a basis for our project. Thus, many of the main components from that guide are present in our architecture model. We had a little trouble designing the architecture model since the creating it on a

functional basis was a foreign concept to us, since we were heavily familiar with the object-oriented paradigm. We decided to create a architecture model which will represent the main components of the application and the functions that were implemented. the function names are semantic and should help understand what component is responsible for what.

Chapter 4

Quality Measures

4.1 Organisational

Typically, a feature is "done" when it covers the defined functional requirement. In some cases where the feature is bigger in scope, we allowed the developers to use their own discretion and experience to break it up into multiple sub-parts that are merge ready. So in bigger features (e.g. User Interface implementation) we allow multiple merges into the main branch without the deletion of the feature branch.

4.2 CI/CD

During the whole development lifecycle we only worked with a documentation pipeline. It is very similar to another pipeline we used for a different project. In short, it automatically builds the document from the LaTeX documentation and delivers this very PDF file as an artifact. Additionally you can trigger a spellchecker manually in the pipeline.

Towards the end of the project we aimed to create a build pipeline for the product itself, where it just builds the project from its files and delivers an executable as an artifact.

4.3 Testing strategy

4.3.1 Frontend

Testing of the front end has been done by just using the application. After implementing the code, we tested the differemt functions and how they interact with the UI. If the desiered behaviour has been reached, the test can be marked as "passed".

4.3.2 Backend

Again, we were fortunate to have worked with the codecrafters challenge, as the most important elements of the backend were implemented with the help of the challenge. Therefore we were able to utilise the built in tests from the challenge to determine if the business logic is correct or faulty. The link to the CLI tool can be found here `https://docs.codecrafters.io/cli/installation`

Part III

Project Documentation

Chapter 5

Initial Project Proposal

Project name: Monad Torrent
MonadTorrent

Team Members

1. Davor Lucic (davor.lucic@ost.ch)
2. Fadil Smajilbasic(fadil.smajilbasic@ost.ch)

Vision

After taking the Functional Programming course we were fascinated by the elegance of Haskell, and how you can get a lot of functionality out of a handful of lines of code. One thing we both never grasped is, how we could actually use that knowledge in real life software projects. Hence, this project will additionally be a learning process for us, where we want to share the knowledge we gained.
The goal and vision of this project is not just a working torrent written in a functional programming language, but it is to show to our fellow students and anyone that comes across this project that building a project with the functional paradigm isn't just possible, but something that should be considered as a valid option next to more common approaches.

Proposed Realisation

As discussed we want to use Haskell to bring this project into fruition. For our frontend there is a Library called "Brick", which will aid us in building a beautiful, but simple Terminal User Interface (TUI). There are many Torrent Libraries available for Haskell, but we plan on staying away from most of them, as we want to get as much of an inside view as possible. We have considered using a torrent file library to handle the file itself, but if possbible, the networking part will be programmed by us completely.

Chapter 6

Project Plan

Describe the project plan as covered in the SEP2 module. A project plan typically consists of the following topics:

- Processes, meetings and roles
- Phases, iterations and milestones
- A **rough** list of things to be done (work items)
- Risk management
- Planning Tools (issue tracker, time tracker, ...)

You should **not** describe your **technical solution** in this chapter. It is all about organizing your project.

6.1 Role distribution and collaboration framework

We have decided to not work with any commonly known method, since it is only going to be two developers working on this application. However, we will use some elements from the SCRUM method, such as weekly meetings to have the opportunity to share our progress and problems over the past week. Nevertheless, we want to encourage spontaneous exchanges between us in form of short conversations between stand-ups or even pair-programming sessions if a light exchange of ideas and inputs doesn't suffice.

We will also use the built-in Kanban board, time tracker, issue board, and milestone board from Gitlab to keep an overview of our progress.

The reason we keep our development method as simple as possible is because we want to spend less time and less resources on managing story points / assigning t-shirt sizes / any other metric and more time building and documenting our product.

6.1.1 Role distribution

To reduce organisational overhead even more, we decided on no fixed roles. The spontaneous coordination of tasks and duties will be possible due to the small size of our team. For this topic again, we want to prioritise developer experience over bureaucracy.

6.2 Project Phases

Unlike in agile methods, where we sprints of fixed time, we want to approach this project with a milestone-based ideology. Milestones can vary in length depending on the size of the scope of said milestone. The milestones for the Project itself are defined as following:

- **Milestone 1 - Initial Setup: 17.09.2024 - 20.09.2024:**Setup repositories and add LaTeXtemplate in documentation repo.
- **Milestone 2 - Research: 17.09.2024 - 30.09.2024:** Define basic conditions for project, such as work methods, organisational topics, functional requirements ann non-functional requirements, risk analysis including risk matrix, actors, rough architecture, and roles.
- **Milestone 3 - Backend MVP: 30.09.2024 - 21.10.2024:** Have a working minimal viable product including their UI elements.
- **Milestone 4 - Expand MVP: 21.10.2024 - 11.11.2024:** With the MVP finished we can start to work on additional features for the torrent client.
- **Milestone 5 - Testing and Integration: 11.11.2024 - 25.11.2024:** Intensifying tests to our product. Fix bugs and improve the torrent as we test with actual users. How we want to approach and search for test subjects will be decided at a later point of the project.
- **Milestone 6 - Documentation and Manual : 25.11.2024 - 09.12.2024:** With the project's ending in sight, we want to start writing the documentation of the products usage in the Gitlab Wiki page, as well as finalising the documentation as much as possible. One big part in the documentation will be the documentation of the architecture. This isn't done before, because the used architecture may change throughout development.
- **Milestone 7 - Project Submission : 09.12.2024 - 16.12.2024:** The last phase will be used for the cleanup of product and documentation, as well as creating the handout necessary for submission as well.

The milestones can also be found on the milestones page of our Gitlab repo.

Chapter 7

Personal Reports

7.1 Davor Lucic

I knew from the beginning it was going to be a challenging project, as my experience with Haskell, the Bittorrent protocol and non-web-based projects was very limited. Even if the project experienced a few speedbumps, I still view it as a success. The beginning was quite smooth, as Fadil and me both got familiar with the Bittorent protocol quite quickly.

The development slowed down quite a bit after that burst of initial productivity, as we introduced the second major component of the application: the user interface. Most of the code example we found online was already outdated, and the documentation provided with the Brick library proved to be lackluster. Thus forcing us to fill in the gaps ourselves and experiment with the library first. After it had already cost us quite a bit of time, we decided to split responsibilities, where I had the honour to explore the UI library. I personally take accountability for not doing enough research about the library. I would have still picked Brick as our tool of choice, since it's still the best documented option out there, but if I knew how painfully outdated most of the sources out there were, I would have invested more time into getting to know the library earlier on.

Personally, working on this project proved to be quite rewarding after all. Even if a bit tedious at times, I've grown to really appreciate Haskell's declarative nature and thus very clean and concise results. My personal highlight was the journey itself. Noticing small improvements each and every day, starting to understand certain error messages more (which the GHC does amazingly well), and seeing everything come together in the end made me feel a grand sense of accomplishment.

7.2 Fadil Smajilbasic

The project was a great learning experience for me. I had never worked neither with Haskell nor with a TUI before, and I was excited to learn a new things about the Haskell language.

I was interested in learning more about the Bittorrent protocol, as I had never worked with it before, just used it because it offers fast transfer speeds.
The usage of the Brick library was a bit challenging at first, as the documentation was not very clear. However, after some experimentation, I was able to get the hang of.
I think the project went well overall. We were able to implement the core functionality of the application, and I am happy with the end result even though we did not implement all the features we wanted to.

Chapter 8

Personal Reports

Before the final submission, **personally reflect** your work in this project:

- What things did go well?
- Which areas could we improve?
- What were your personal highlights?

The information gathered in this chapter will be **very** useful for all your future projects.

8.1 Davor Lucic

I knew from the beginning it was going to be a challenging project, as my experience with Haskell, the Bittorrent protocol and non-web-based projects was very limited. Even if the project experienced a few speedbumps, I still view it as a success. The beginning was quite smooth, as Fadil and me both got familiar with the Bittorent protocol quite quickly.

The development slowed down quite a bit after that burst of intial productivity, as we introduced the second major component of the application: the user interface. Most of the code example we found online was already outdated, and the documentation provided with the Brick library proved to be lackluster. Thus forcing us to fill in the gaps ourselves and experiment with the library first. After it had already cost us quite a bit of time, we decided to split responsibilities, where I had the honour to explore the UI library. I personally take accountability for not doing enough research about the library. I would have still picked Brick as our tool of choice, since it's still the best documented option out there, but if I knew how painfully outdated most of the sources out there were, I would have invested more time into getting to know the library earlier on.

Personally, working on this project proved to be quite rewarding afterall. Even if a bit tedious at times, I've grown to really appreciate Haskell's declarative nature and thus

very clean and concise results. My personal highlight was the journey itself. Noticing small improvements each and every day, starting to understand certain error messages more (which the GHC does amazingly well), and seeing everything come together in the end made me feel a grand sense of accomplishment.

Chapter 9

Regular Meetings

The Monad-Torrent tean met on a regular basis with their professor and project supervisor. The dates were determined on a rather spontaneous basis, although rarely surpassing more than 10 days between meetings.

Meeting Agendas were prepared at the end of each meeting for the next meeting. These agendas were of fixed structure in the beginning, as we had to coordinate formalities around the projects direction and the actual goals. With time they became more fluent, as product development and product demos became a focus the more time passed.

Since we are a team of two developers, detailed meeting minutes were of second priority, as we only had the meetings with all members present. A short bullet-point list was created and shared after each meeting with the tasks-to-be-done. This philosophy helped us further achieve the goal of prioritising developer satisfaction over creating redundant bureaucracy.