Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: rendt

Analysis Report

Project Group Members:
Huseyn Allahyarov 21503572
Mahammad Shirinov 21603176
Ibrahim Mammadov 21603109
Cenk Er 21600937
Nurlan Farzaliyev 21503756

Supervisor: İbrahim Körpeoğlu
Jury Members: Hamdi Dibeklioğlu and Özcan Öztürk

Analysis Report
November 11, 2019
This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

Analysis Report

Project Short-Name: rendt

1 Introduction

With the ever-growing advancements in computer algorithms, machine learning tools, and with the availability and accessibility of such tools, distributed computing and cloud computing systems have become extremely widespread, to the point of almost being a necessity. [1][2]

The biggest players in these fields are currently Amazon (AWS), Microsoft (Azure) and Google (Google Cloud). What's common among these providers is that they all have large centralized networks of nodes somewhere in a server farm (or several farms), and they provide users with computing power and other services such as storage using parts of that network.

The fact that all the nodes belong to one party and are hosted in one or a few dedicated locations gives them great reliability and ability to offer good pricing. However, since there are few providers and millions of users, only the few big players make revenue. Also, the amount of computing power that is available is dictated by the sole provider.

With Rendt we propose an alternative solution to this vast need of distributed and cloud computing power, where multiple parties can offer their machines and get paid in return, while users will still have access to the computing power that they need for their projects and experiments. Similar solutions have been used in the academic community to solve problems like the decomposition of natural numbers as a sum of three cubes, [3] but they are voluntary in nature and have no commercial use. [4]

1.1 Description

Even with all this need for computing power, there is still a large amount of idle or underused machines, which are mostly computers that belong to personal users or small companies that don't have distributed computing infrastructure. Rendt is a platform where these idle resources can be shared with people who need them and turned into profit. Users can choose a resource on their machine that they want to share, choose a type of task they are willing to share it for, and possibly state their price. Then, other users in need of that particular resource will see different offers for their query, choose the one they like and start using resources. After the tasks are run and results sent back to the task issuer, the payment will be made and transaction will be completed.

Using Rendt, somebody with powerful GPUs in their laptop who may not need them for personal use, at least not all the time, can turn it into profit, and dually, somebody who needs GPUs to run get some work done (train neural networks) but doesn't have access to any can look for people offering their resources and have access to them without owning any GPUs. Rendt is going to play the role of a match-maker and regulator in such transactions. It will host users that share their resources, along with detailed information about their computing power, types and availability of resources and other information, for example, the time the person is willing to lend their resources or the history of the tasks they have successfully (or unsuccessfully) completed.

Rendt will be used for remotely running different kinds of tasks, such as training neural networks, rendering videos, or plain mathematical (CPU-heavy) calculations. This compartmentalization will help users better find suitable offers, since machines best suited for a particular kind of job will be tagged by that category by the host themselves. Additionally, Rendt will support distributed computations as well: if the user's task is parallelizable, they will be able to request more than one node for their task and run it on them and then receive the single result back, as if running their task on some centralized cluster.

When a user issues a task and selects a node(s) to run it on (or leaves the selection to the system), the task goes to our central server, where a match is registered, and the transfer of the needed files (data, binaries, scripts) is initiated (e.g. by passing the files from issuer to the central server and from there to the host, by P2P file transfer etc.). At this stage, a part (or the whole) of the payment the issuer has agreed to pay is deducted from their account and held by Rendt. The host node(s) start running the task(s), and once done, transfer the result files back to the issuer. Then, the host receives the payment and the transaction is over.

2 Current System

To start with, we researched about similar projects to see if such a system is feasible. We came across BOINC which is a system dedicated to volunteering CPU cycles for scientific research for institutions such as SETI - Search for extraterrestrial intelligence. It is used by research institutions to use volunteers' computers for research purposes. However, it is limited in the sense that only a few privileged users can run jobs, and the volunteers don't profit from this interaction.

With this research and the suggestions from our innovation expert, we started to research the different technologies to implement such a system. First, we started to gather information about WebAssembly which is a web-based technology that our innovation expert suggested us to look into. WebAssembly is a fairly new technology that allows codes written in high level languages like C/C++, Rust to be compiled into web. With WebAssembly we could implement a system that doesn't require any installation and supports all the platforms. However, with extensive research into the matter, it turned out WebAssembly would not be a good choice for such a system. The main reason for this is that WebAssembly uses Enscripten under the hood which converts the code written in high level languages to vanilla javascript. However, Enscripten is not able to convert all the code natively and performance decrease is to be expected because of its usage of web workers rather than cpu threads. Finally, WebAssembly and Enscripten cannot natively support multithreaded code and low level libraries such as OpenMPI is not supported by this technology.

Recently we came to a conclusion that a program written in a high level language is mandatory and it should run cross-platform with native performance and unfortunately, download and installation is necessary. With all this research, we finally settled for implementation with C++ that runs cross-platform with the help of CMake.

3 Proposed System

This section includes details of the proposed system and its details for both users and development.

3.1 Overview

Rendt is aimed to be a cross-platform desktop application that runs in the background without the interference of rented PC's user. The project should run natively to avoid compromising the performance and security should be provided to protect the rented PC from any malicious code or users. Security is an ongoing research mostly aimed at virtualization technologies. Another point to be considered is the secure transfer, compilation and execution of the tasks without giving out any details about the task to the renter to provide confidentiality. Rendt's main scope is general code execution at the moment. Code execution may be very power demanding depending on the type of the task to be executed. For tasks such as image analysis and computer vision, code execution depends solely on the hardware and performance of the CPU and, in some specific cases, GPU and execution time is inversely proportional with the hardware performance. Users, especially renters, should be informed of the drawbacks of performance rental and the decrease in CPU and GPU lifetime as a consequence as well as the responsibility of renting. Secure and safe payment and refund systems should be provided for both renters and users. Price of rentals should be calculated in terms of the hardware and performance. Renters will be rated in terms of reliability, in other words, if a renter does not take the

responsibility of renting and fails to finish execution of the task, a bad reputation is to be expected. Rendt's users scope is expected to be large and unlimited and to accomplish this goal, the system should be straightforward to use and technical knowledge of the user and requirements should be minimal.

3.2 Functional Requirements

This section includes requirements for the key features of the system that a user will be interacting with.

3.2.1 Registration

Users should be registered to use Rendt. With the user accounts, a proper rating system can be implemented to display reliability rating of the user, hardware details, past experience with Rendt (in terms of rentals) and other users can distinguish the renters via their accounts. To be able to use the system, users should log into their accounts if they have already registered, otherwise register and then log in.

3.2.2 Main menu

This is the first screen that a logged in user interacts with. Users will be selecting if they want to rent their PCs or use available rental PCs. If a user wants to use the system, he/she will be displayed a list of available PCs with the hardware details, hourly price and rating. Users may visit the renters' accounts to display information about the user and reviews. If a user wants to rent his/her PC, a fair price will be calculated and suggested for the current hardware, however user is responsible of setting the price. Renter will also set a duration for the rental to provide himself/herself freedom to stop renting after certain time.

3.2.3 Selection and accepting

When a user is selecting the rental system for use, it is his/her own responsibility to choose the right system and user for the execution. If a task takes more time than the renter's duration, renter has the right to finish executing halfway. In other words, if a rental PC is available for 2 hours and user's execution takes more than 2 hours, renter can wait for the execution or stop it without getting penalized.

Moreover, after user selects the PC to be rented, a notification/request will be sent to the renter. Renter may or may not accept the request. The execution will start after accepting. Another notification will be sent to the user about the response of renter.

3.2.4 Execution process

After selection and acception stage, user will be granted to use the rental PC for execution of the task(s). Via a drag & drop screen, user will send the files for the execution and status of the execution and file transfer will be displayed for both the user and renter. User will be able to stop the rental and after he/she is done with it. Renter will be notified as well.

3.2.5 Payment

For a secure payment, payments will be sent when the PC is rented and execution started, but the renter will receive the payment once execution and rental is finished. This system is to avoid misuse of the system. Via this payment system we aim to reinforce the responsibility of both sides. An unfinished renting job will not get paid and the user's payment will be refunded. To avoid misinformation, the system will decide if the task executed correctly and completely or the execution failed due to a technical failure.

3.2.6 Reliability Rating

Users' reliability rating points will be decreased/increased depending on the execution status of the task. If a task is not executed fully or failed due to a technical issue such as shutting down the system, system will lower the rating of the renter.

3.3 Non-functional Requirements

This section includes requirements for the developers and development process.

3.3.1 Performance

To provide native performance, the system will be implemented with a high level language such as C++ to execute the tasks natively without any performance compromises. Virtualization is expected to be used to provide security, but should not decrease performance. With wide virtualization support in majority of CPUs, performance drop should be the case.

3.3.2 Compatibility

Rendt should be a cross-platform application to support a wide variety of systems and OSs. To tackle this issue, we intend to use CMake with C++ which can compile a C++ program to run on Windows, MacOS and Linux systems. With the compatibility provided, rendt will be available for the use of a wide variety of users with different hardware, OS and configuration.

3.3.3 Maintainability

Rendt will utilize a server to serve the users and renters. Server will be responsible from payment, reliability rating decision, rental requests and responses, file transfer and displaying available PCs. Server will be maintained by us. Maintenance covers server rental payment, network issues with the server and providing the system online.

3.3.4 Security

Rendt will be providing secure payment system, execution process and execution environment. To provide a secure payment system, money transfer will be sent and hold until the execution ends. Renter will get paid only after the execution finished successfully. To provide a secure execution process, rendt will operate in the background without giving away any information about the task to provide confidentiality for the user. Rendt will be operating via virtual machine (VM) or a technology alike to provide a secure execution environment. VM will provide the necessary security to avoid getting attacked in the renter's end. Any damage will be happening in the virtual environment.

3.3.5 User interface

As mentioned above, UI should be straightforward and a user without an advanced level of technical knowledge of computers should be able to operate the rendt app and be able to use it for renting or using rental PCs. To provide such a straightforward system, VM initialization and installation should be handled by either the rendt installation or with some automated or manual process. This is an ongoing process.

3.3.6 Scalability

System should be able to serve as many requests as possible. With the increasing number of requests and users, system should be able to serve without any compromises. To tackle this issue system should be able to serve with minimal resource allocation per request.

3.3.7 Reliability

Rendt should be a reliable source for the users that need better performing hardware. For the rendt's side, server should be up and running, serve the requests as necessary, provide the security, scalability and compatibility. For the users/renters' side, user reliability rating should be considered before rentals alongside the reviews from the past experiences of the user.

3.4 Pseudo Requirements

This section will provide specific information about the development process.

3.4.1 Implementation

Rendt should be implemented with a high level language such as C/C++ to provide native performance and take advantage of multithreaded code and, if possible, distributed execution. External libraries are expected to be used for the compatibility, user interface and performance.

3.4.2 DevOps tools

DevOps tools will be utilized for the development process. These tools will include JIRA for task assignments and issue handling, GitHub for the version control, Slack for communication and etc. Usage of these tools will help us in the long-term for tracking changes, handling changes, track the development progress, task distribution etc.

3.4.3 Design paradigm

Rendt will be developed with Object Oriented design pattern. OOP is the most comfortable implementation pattern for all of rendt developers and a reliable choice for the implementation.

3.4.4 Data encryption

All the data residing in the system and the server should be encrypted to provide necessary security. User information, account details and payment details should be kept encrypted in the system.

3.4.5 Testing

After each milestone in the development process, an advanced test should be executed before starting to work on the next work package. These tests will be simulated on the scenarios and boundary cases as well as rare but possible cases. Such process should ensure reliability before going too deep into the other parts of the project to avoid uncertainties.

3.5 System Models

3.5.1 Scenarios

3.5.1.1 Sign Up

Primary Actor: New user

Interests: 1. A new user wants to create an account

Entry conditions: 1. User has the GUI application installed on their machine

Exit conditions: 1. User has created an account

Success scenario event flow:

1. User open the application
2. User clicks the Sign Up button
3. User fills in the required information and submits form
4. User confirms his email address

3.5.1.2 Login

Primary Actor: An existing user

Interests: 1. User wants to log in to their account

Pre-conditions: 1. User has created an account and has their credentials

Entry conditions: 1. User has the GUI application installed on their machine

Exit conditions: 1. User has logged into his account

Success scenario event flow:

1. User open the application
2. User enters his credentials and clicks the Sign in button

Alternative event flow:

1. User open the application
2. User selects the Forgot Password option
3. User follows the link in his email to restore his password

3.5.1.3 Leasing hardware: post availability of your hardware

Primary Actor: User who wants to share his computational power (receiver)

Interests: 1. User wants to lease their hardware

Pre-conditions: 1. User has rendt application installed and has created an account

Exit conditions: 1. Receiver's post is submitted the dashboard

Success scenario event flow:

1. User opens hardware leasing form
2. User fills in the details about the lease (time, number of cores etc.)
3. User submits the form
4. Rendt server receives the form and posts receiver's details to the global dashboard
5. Receiver gets a confirmation message

3.5.1.4 Accept job

Primary Actor: User who wants to share his computational power (receiver)

Interests: 1. User wants to lease their hardware and earn money

Pre-conditions: 1. User has rendt application installed and has created an account

Entry conditions: 1. Receiver receives the request

Exit conditions: 1. Receiver accepts the request

Success scenario event flow:

1. Receiver receives a request of a task
2. Receiver accepts the execution of the task
3. Rendt begins the execution of the task on receiver's computer
4. The task is finished or the allocated time is used up
5. Rendt sends the results of the execution to the sender and cleans receiver's environment
6. Receiver receives his payment in exchange of this transaction

Alternative event flow:

1. Receiver receives a request of a task
2. Receiver denies the request.

3.5.1.5 Rent Hardware

Primary Actor: User who wants to rent computational power (*sender*)

Interests: 1. User wants to rent hardware and run their jobs

Pre-conditions: 1. User has rendt application installed and has created an account

Exit conditions: 1. User submits their job to a node (nodes)

Success scenario event flow:

1. User opens the dashboard and uses filters to find node(s) he can use
2. User selects a node (nodes)
3. Rendt server sends requests to all the nodes
4. Nodes accept the request
5. Rendt initiates the execution of the task(s)
6. After tasks are executed and finished, they are collected back in the rendt server
7. Rendt server unifies the results and sends them to the sender
8. Paymen system coordinates the due payments

Alternative event flow:

1. User opens the dashboard and uses filters to find node(s) he can use
2. User selects a node (nodes)
3. Rendt server sends requests to all the nodes
4. Some nodes reject the request
5. User starts over from step 1

3.5.2 Use-Case Model

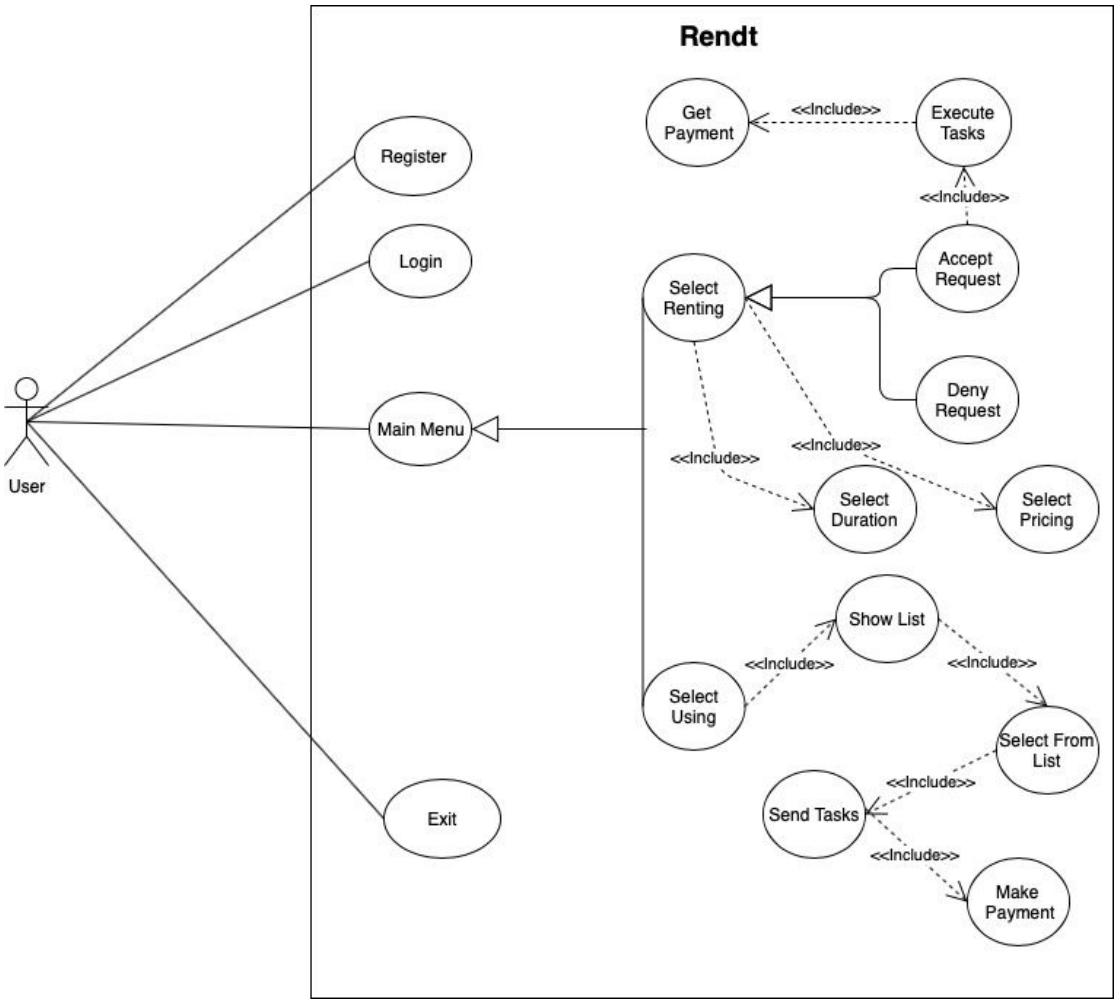


Figure 1. Use-Case diagram

3.5.3 Object and Class Model

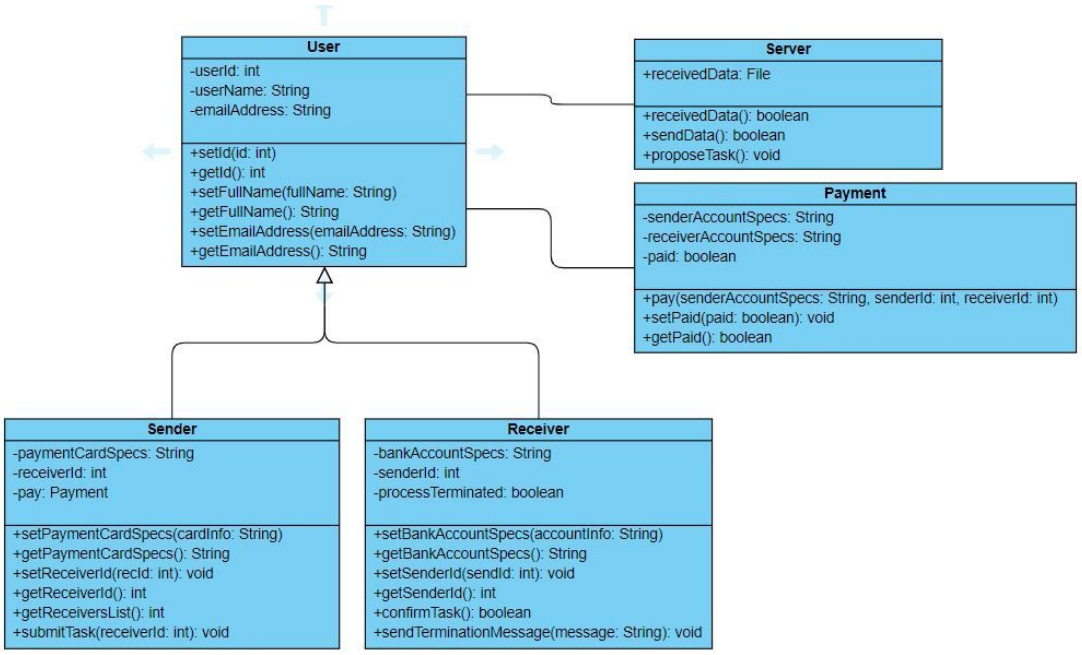


Figure 2. Class model

3.5.4 Dynamic Models

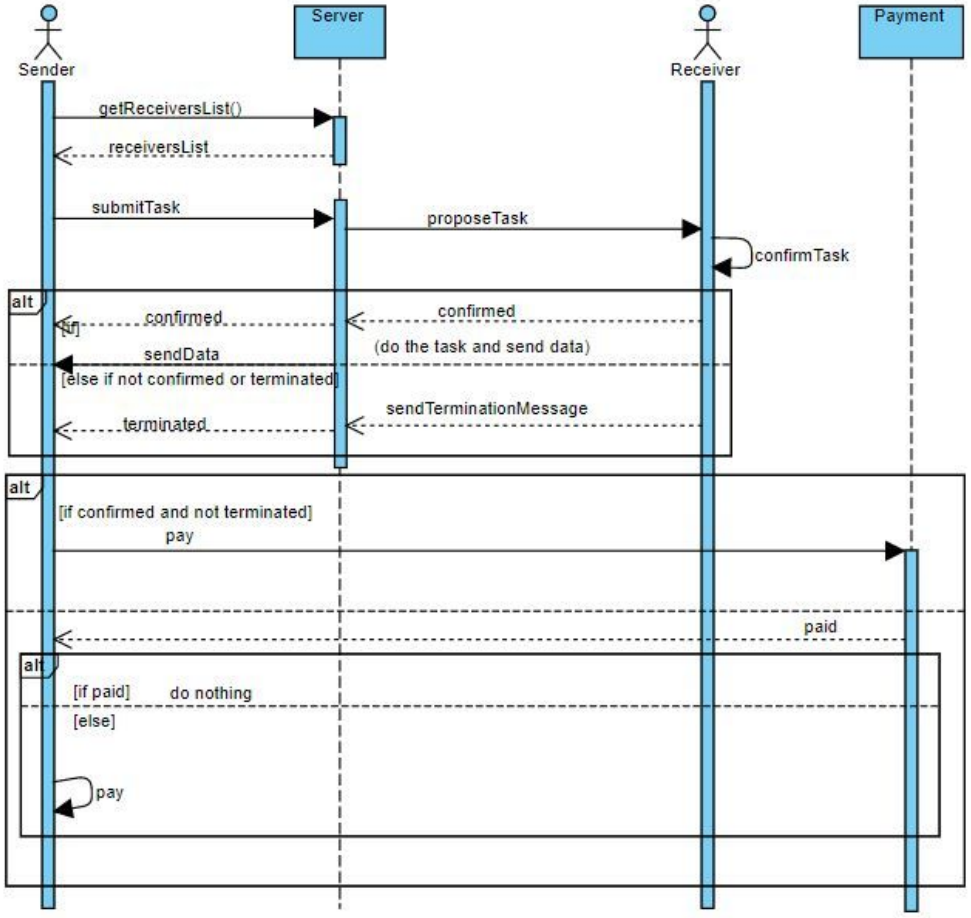


Figure 3. Sequence diagram

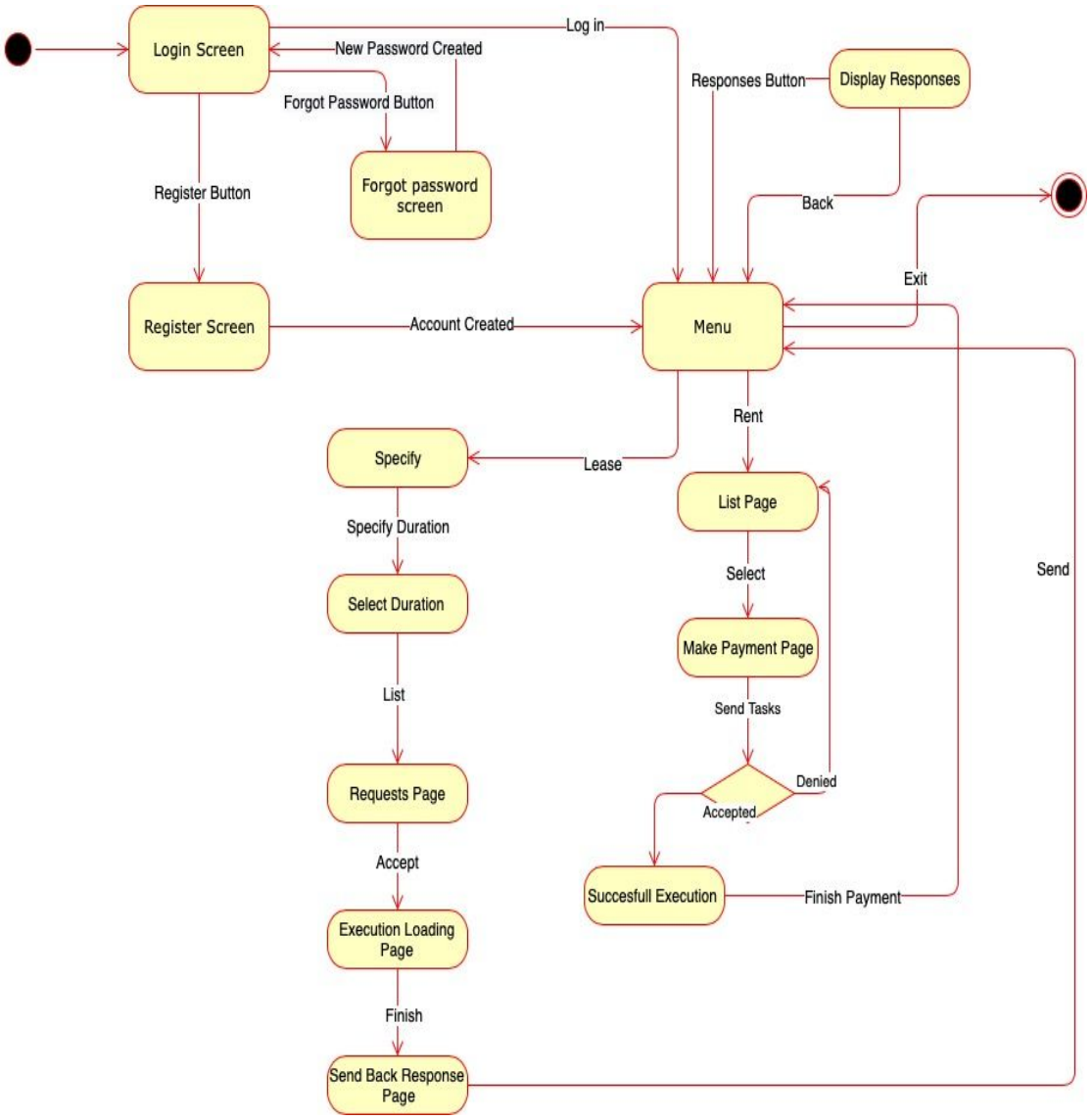


Figure 4. State diagram

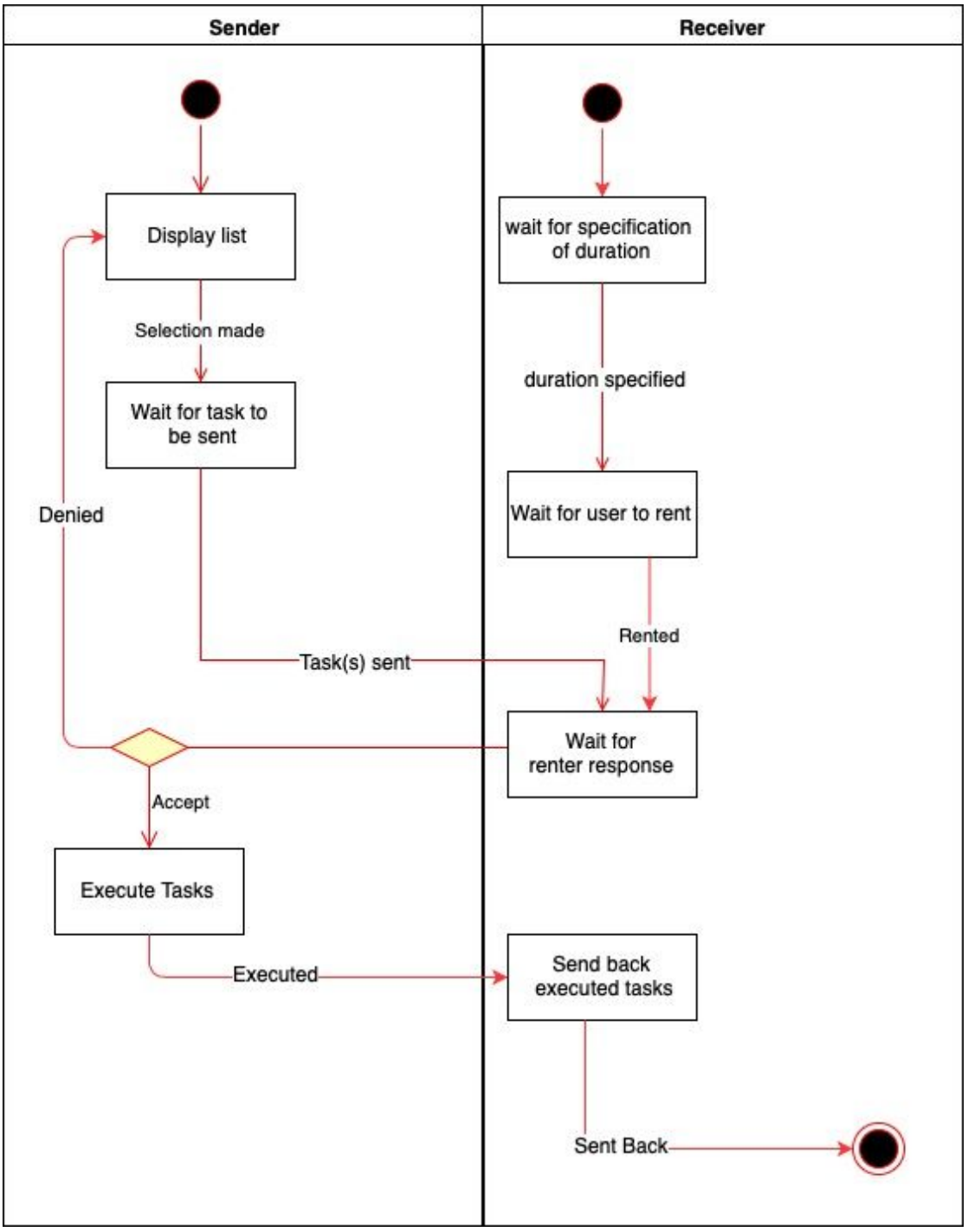


Figure 5. Activity diagram

3.5.5 User Interface

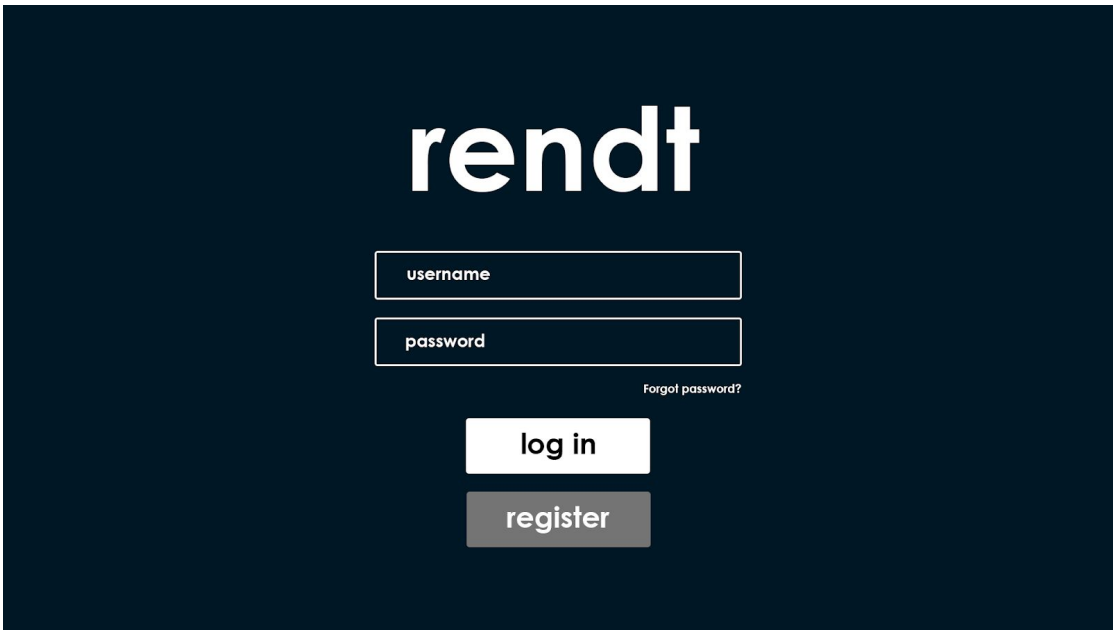


Figure 6. Log-in Screen

This is the first screen that welcomes the user. User, if already registered, needs to log in to the system by entering his/her username and password. If user has not registered yet, he/she can register by clicking the register button. If a user has forgotten the password, he/she can click "forgot password?" button and an email will be sent to the user.

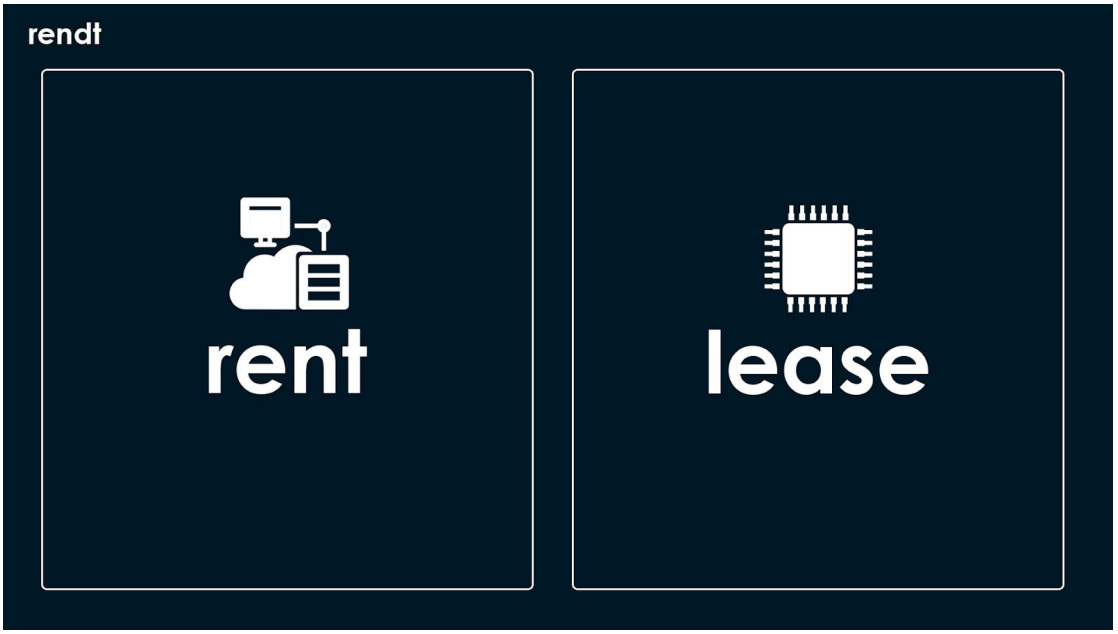


Figure 7. Renter/Leaser selection

This is the screen that user will be entering after logging into the system where he/she needs to select whether he/she wants to rent other available PCs or lease their own.

Figure 8. Leasing Configuration

If user selects the **lease** option, a new screen opens up for hardware information and configuration (where user may set how much hardware to be leased) and duration for leasing (in terms of date and time).

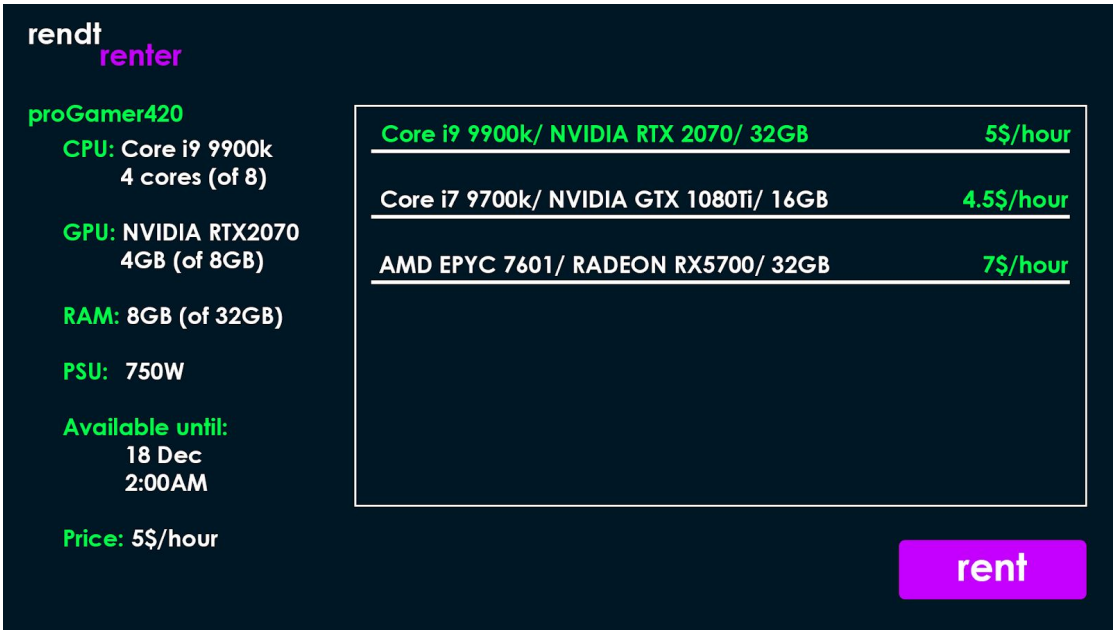


Figure 9. Available PCs list

A user that wants to rent other available PCs (renter) will be displayed a list of available PCs with necessary information including hardware configuration and availability duration.

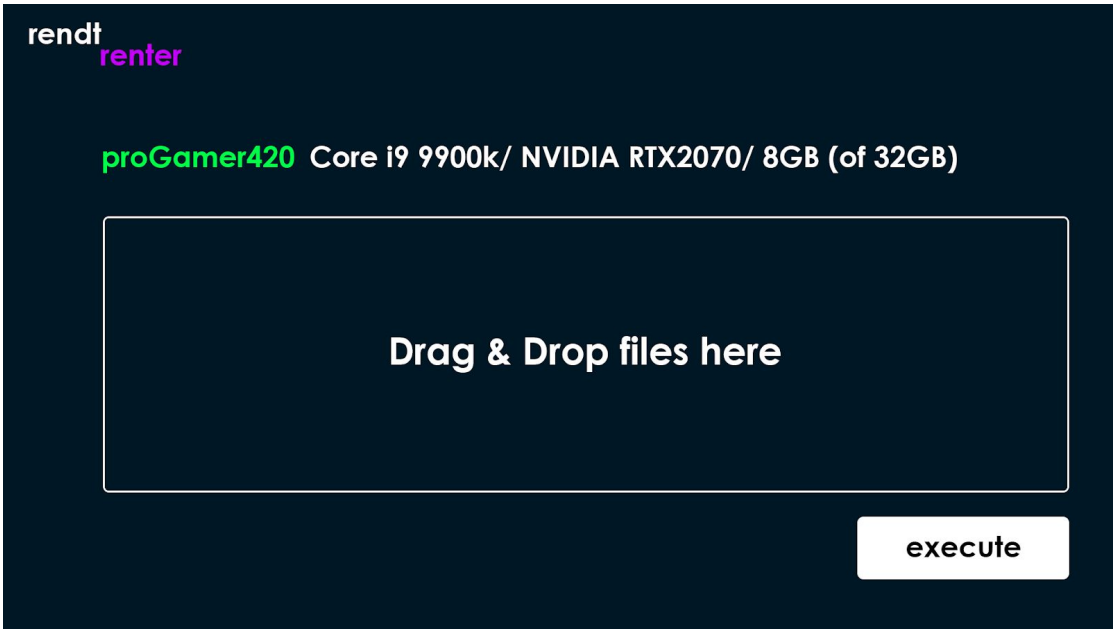


Figure 10. File upload screen

After selecting the PC to rent, and leaser accepts the request, a new window will be shown where renter will upload the files to be executed.

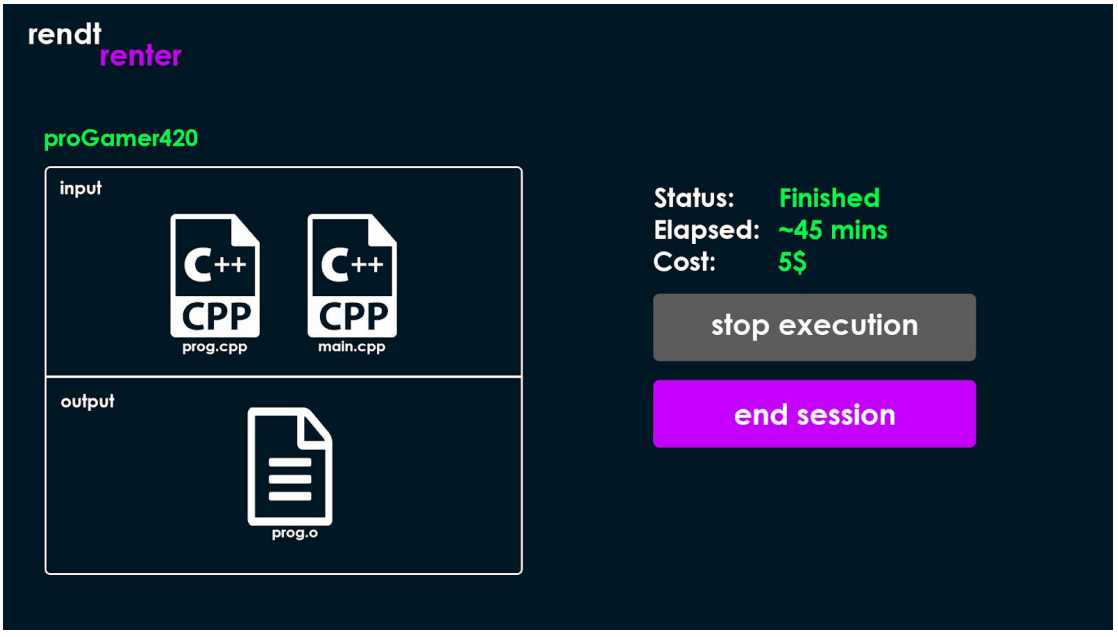


Figure 11. File execution screen

After starting the execution, input and output files, execution status will be shown to both leaser and renter. Renter has the option to stop execution when necessary. When the leaser is done with the execution, he/she will end the session and the payment will be sent to the leaser and the leased PC will be listed again.

4 Other Analysis Elements

4.1 Consideration of Various Factors

Apart from all the factors above, it is important to take into account some elements outside of the scope of development process. Our project, although heavily dependent on systems development, has some profound effects on and restrictions with regards to factors such as public safety, welfare, environment and economy.

One of the most important and restricting factors of our project is public safety - the security implications of our system (both personal and global), and privacy. Since we administer the execution of one person's code on another user's machine, care must be taken to ensure the privacy of receiver user's files and protection of their system. On the other side of the coin, the receiving user should also have no access to sending user's code and files, which might be intellectual property. One other concern could be the execution of malicious attacks with the aid of our system, such as launch of DDOS attacks after being granted access to a number of machines.

Another major environmental factor that actually inspired this project idea is concerned with leveraging idle computing power throughout the world. Most computers aren't actively used most of the time, so this computing power, instead of sitting idle, can be provided to people that are willing to pay to use it. This would result in less hardware being underused, and thus, less unnecessary hardware production/consumption. Also, people in possession of idle computers would be able to profit from them, contributing to their welfare.

The economic implications of this project could also go beyond personal users leveraging their hardware and turning it into profit. It can also help a group of people, such as friends or a team in a company, establish a cluster among them with low costs and share their resources freely between them.

Table 1. Various Factors

	Effect level	Effect
Public safety	8	Security of receiving user's system and privacy of their files. Privacy of sender's code and data. Risk of misuse of our system, e.g. DDOS attacks
Environmental factors	7	Idle computing power utilized; demand for new hardware reduced.
Public welfare	5	Profit from idle computers.
Economic factors	5	More accessible clusters for companies and teams.
Public health	0	N/A
Cultural factors	0	N/A
Social factors	0	N/A

4.2 Risks and Alternatives

Some of the risks that we might face during our development are the ones concerning efficiency and user convenience. For one, we might discover midway in our development that the usage of virtual environments for security has profoundly negative effects on the code execution performance, making it completely unusable. We have done some preliminary research and have a high confidence that such a scenario is unlikely, but in the event that it happens, we will stop executing senders' code in virtual environments and run

them directly in native operating system of the receivers. To provide some level of security, we will incorporate methods for detecting malicious users and specifically malicious pieces of code, so as to protect the receiving users' machines. We will also provide the receivers with a manual of how they can maximally protect themselves (e.g. receive tasks only when they are logged in as a non-admin user which does not contain any of their personal files).

Another unforeseen difficulty may arise if the distributed execution of tasks turns out to be extremely slow due to the inherent structure of our system (communication done over the internet, rather than fast wired connections). In this case, we will shift our focus from supporting distributed execution to P2P (one sender and one receiver) execution.

Another much less detrimental risk is concerned with user convenience. Before a receiving user can receive and run any task, a virtual environment must be installed on their machine. If during the development process we discover that the automatic/remote installation of such a virtual environment by us is impossible or very hard, we will provide the users with a step-by-step instruction manual so that they can set the environment up themselves, with minimal to no tech knowledge.

Table 2. Risks

	Likelihood	Effect on the project	B Plan Summary
Virtual environment inefficiency	<5%	The code delivered to the user could be malicious.	Use malicious code detector and inform users of security risks and practices.
Distributed processing unscalability	<10%	No distributed execution.	Focus on P2P computation.
Inability to set up virtual environment remotely	50%	Users setup their environment themselves.	Provide users with a comprehensive instruction manual.

4.3 Project Plan

We have identified some milestones and divided our project into subtasks. Before any development, we are planning to prepare a high and low level design of our project, to set out a roadmap for software architecture. Then, we will start the development, and have in mind the following milestones: the **initial working backbone** of the system where one user can send a job to one other user and have it executed, **central server** which will serve as the delegator in all job transactions and regulate the sending/receiving of the jobs, **virtual environment** software where we will coordinate the execution of all tasks in a safe sandbox environment to provide security, the update of the backbone to **incorporate parallel tasks** on different receiver machines, final **GUI** for all users to interact with the system and finally a **payment system** for all the transactions.

Table 3: List of work packages

WP#	Work package title	Leader	Members involved
WP1	High Level Design	Mahammad Shirinov	Huseyn Allahyarov Nurlan Farzalyev Cenk Er Ibrahim Mammadov
WP2	Low Level Design	Huseyn Allahyarov	Mahammad Shirinov Nurlan Farzalyev Cenk Er Ibrahim Mammadov
WP3	P2P Task Execution	Huseyn Allahyarov	Mahammad Shirinov
WP4	Central Server Configuration	Cenk Er	Huseyn Allahyarov Mahammad Shirinov

WP5	Virtual Environment Setup	Ibrahim Mammadov	Nurlan Farzaliyev
WP6	Distributed Task Execution	Mahammad Shirinov	Huseyn Allahyarov Cenk Er
WP7	Build a GUI	Ibrahim Mammadov	Nurlan Farzaliyev
WP8	Payment System	Nurlan Farzaliyev	Ibrahim Mammadov Cenk Er

Table 4. Work Packages in detail

WP 1: *High Level Design*			
Start date: *11.11.2019* **End date:** *31.12.2019*			
Leader:	Mahammad Shirinov	**Members involved:**	Huseyn Allahyarov Nurlan Farzalyev Cenk Er Ibrahim Mammadov
Objectives: Define design goals and construct the architecture of the software system to be built. Identify smaller subsystems that can be realized by individual subgroups.			
Tasks: ***Task 1.1 Writing report*** **:** *To come up with Subsystem decomposition, hardware and software mapping with access control and security and so on.*			
Deliverables ***D1.1:*** *High Level Design Report*			
WP 2: *Low Level Design*			
Start date: *03.02.2020* **End date:** *17.02.2020*			
Leader:	Huseyn Allahyarov	**Members involved:**	Mahammad Shirinov Nurlan Farzalyev Cenk Er Ibrahim Mammadov
Objectives: Through Low Level Design, we close the gap between the application objects and the off-the-shelf components by identifying additional solution objects and refining existing objects			
Tasks: ***Task 2.1 Writing report:*** *Define object design trade-offs, interface guidelines and engineering standards.*			
Deliverables ***D2.1:*** *Low Level Design Report*			
WP 3: *P2P Task Execution*			
Start date: *18.02.2020* **End date:** *06.03.2020*			
Leader:	Huseyn Allahyarov	**Members involved:**	Mahammad Shirinov
Objectives: Setup the delegation of a task between 2 users in P2P fashion, see what could be done and how, create a basic message passing interface.			
Tasks: ***Task 3.1 P2P system basics*** **:** *Learn and understand the basic principles of P2P system* ***Task 3.2 Message passing interface*** **:** *Establish the communication between 2 users.*			
Deliverables ***D3.1:*** *Basic backbone of the project - system configured in P2P fashion.*			
WP 4: *Central Server Configuration*			
Start date: *06.03.2020* **End date:** *26.03.2020*			
Leader:	Cenk Er	**Members involved:**	Huseyn Allahyarov Mahammad Shirinov
Objectives: *Setup and configure the central server to act as a delegator between task senders and receivers.*			
Tasks: ***Task 4.1 Server creation*** **:** *Create the server and set up its receiving and outgoing ports.* ***Task 4.2 Sender-server communication*** **:** *Establish communication between the server and sender. The server should successfully receive a task from the sender and store it, to further delegate to a receiver to be run.* ***Task 4.3 Server-receiver communication*** **:** *Establish communication between the server and receiver. The server should successfully send a task to the receiver and receive back its results.*			
Deliverables ***D4.1:*** *The central server application.*			
WP 5: *Virtual Environment Setup*			

Start date: *06.03.2020* **End date:** *10.04.2020*			
Leader:	Ibrahim Mammadov	**Members involved:**	Nurlan Farzaliyev
Objectives: *Create a virtual environment for safe execution.*			
Tasks: ***Task 5.1 Virtualization basics*** **:** *Learn the basics of virtualization and create it on the computer.* ***Task 5.2 Virtual environment creation*** **:** *Create a virtual environment on other computer using the P2P system and Server created for that task.* ***Task 5.3 Code execution*** **:** *Try to execute code in a virtual environment. The code written on one computer will be delivered to other computer and executed in its virtual environment.*			
Deliverables ***D5.1:*** *Virtual environment application.*			
WP 6: *Distributed Task Execution*			
Start date: *26.03.2020* **End date:** *30.04.2020*			
Leader:	Mahammad Shirinov	**Members involved:**	Huseyn Allahyarov Cenk Er
Objectives: *Realize the execution of a task submitted by one sender on several receivers' computers.*			
Tasks: ***Task 6.1 Task distribution*****:** *update our server to be able to send tasks to multiple receivers.* ***Task 6.2 Result collection*****:** *update our server to be able to reconcile results from multiple receivers.*			
Deliverables ***D6.1:*** *Updated fully functioning version of the core of our system*			
WP 7: *Build a GUI*			
Start date: *10.04.2020* **End date:** *22.04.2020*			
Leader:	Ibrahim Mammadov	**Members involved:**	Nurlan Farzaliyev
Objectives: *Create well organized, user friendly and attractive interface of application*			
Tasks: ***Task 7.1 User friendly interface*** **:** *main task of this work package is to come with extremely user friendly interface so that even non technical users would be able to use the application* ***Task 7.2 Sender GUI*** **:** *Sender should come up with the sender window where he should be able to select sending files, press send button and so on* ***Task 7.3 Receiver GUI*** **:** *The receiver should get the window where he accepts the communication if his device is free to use.*			
Deliverables ***D7.1:*** *GUI*			
WP 8: *Payment System*			
Start date: *22.04.2020* **End date:** *30.04.2020*			
Leader:	Nurlan Farzaliyev	**Members involved:**	Ibrahim Mammadov Cenk Er
Objectives: *Create safe and secure payment system*			
Tasks: ***Task 8.1 Learning and investigation*** **:** *Investigate the shortcuts for creation of payment systems* ***Task 8.2 Implementation of payment system*** **:** *Implement the system with all considerations of security and safety*			
Deliverables ***D8.1:*** *Payment system*			

Figure 12. Gantt chart part 1

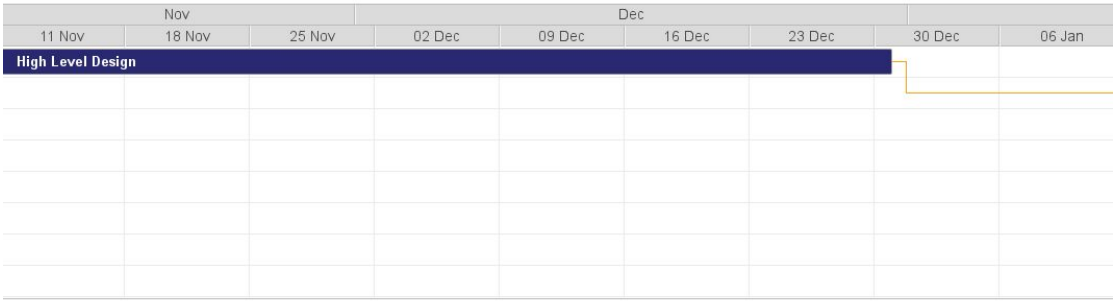
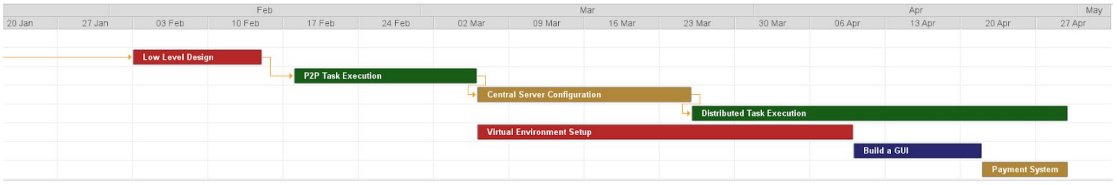


Figure 12. Gantt chart part 2



4.4 Ensuring Proper Team-Work

We are planning to make use of Jira Software for collaboration and management of our project, and to communicate and address our challenges through this platform. There are a plethora of different variations of such tools, and a brief research of these tools showed that Jira is the solution that has all major features that we need, and is free to use.

For version control and code hosting we will use Github. The contribution of each member in terms of lines of code will be automatically tracked by Github. But the number of lines of code is not a perfect indicator of work done. To ensure that every team member makes an equal amount of contribution to the project and gains an equal amount of knowledge and experience from it, we have carefully distributed the subtasks of our projects to different members. Every team member will take lead in some subtask, and see it completed.

4.5 Ethics and Professional Responsibilities

In view of all the factors in **4.1** and many others, we have identified some important responsibilities that need to be addressed. First, we have decided to try our best to ensure the safety of users of our system on either side, be it the security of their system or privacy of their data. This is very important, and we are willing to sacrifice some efficiency of our system and considerable development time to address this issue.

One other ethical concern is the fairness of the payment system. Wherever there is monetary payment involved, things must work precisely, and we want our payment system to also be foolproof. What's more, we need to address some boundary cases and have policies prepared for different scenarios; for example, if the task is interrupted midway in its execution and never finished, does there have to be any payment?

Finally, we need to sustain a certain level of reliability of users in our system. There could be users (receivers) that have a very low task completion rate; they take on some process, but it rarely terminates and sends back the results due to either their interruption or factors out of their control, like electricity/internet shortage. We should make sure to keep such users identifiable. To this end, we are planning to record the task completion rate of receivers and have a reliability score associated to them based on this rate that anyone will see before submitting their task.

4.6 New Knowledge and Learning Strategies

Our project is a very vast and complicated one and there are a lot that we should investigate and learn. The most important thing is that we need to know how to implement distributed systems and which technologies we should use for that. We are planning to implement the core of our project using C++ and for distributed part we are planning to use MPI(message passing interface) library. For virtualization we should learn how and with which technologies we can create a safe and secure virtual environment. At the same time we should gain some knowledge about cloud computing, network issues and how to build and manipulate servers. We also are going to have some kind of payment system in our project, so we should investigate about the implementation of money transaction systems. Although we all worked and have experience in C++ we still should consider and learn many implementation details of this language like CMake for example. The strategy for learning we are planning to apply is very straightforward. We are planning to watch

some tutorials about distributed systems, cloud computing, C++ development and so on. Also one of our group members took Parallel computing lesson and have some knowledge about distributed systems, another member of our group is taking Network course which we think will also be helpful and for the next semester some of us consider taking distributed systems course which is the core of o

5 References

[1] "What is distributed computing and what's driving its adoption?". Packt. https://hub.packtpub.com/what-is-distributed-computing-and-whats-driving-its-adoption/ (accessed October 14, 2019).

[2] "The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle". Dr. Brijender Kahanwal, Dr. T. P. Singh. International Journal of Latest Research in Science and Technology, Vol. 1, No. 2, pp. 183-187, 2012

[3] University of Bristol. "Sum of three cubes for 42 finally solved -- using real life planetary computer." ScienceDaily. www.sciencedaily.com/releases/2019/09/190906134011.htm (accessed October 14, 2019).

[4] "Seven Ways to Donate Your Computer's Unused Processing Power". Vice. https://www.vice.com/en_us/article/bmj9jv/7-ways-to-donate-your-computers-unused-processing-power (accessed October 14, 2019).