



Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: rendt

Final Report

Project Group Members:

Huseyn Allahyarov	21503572
Mahammad Shirinov	21603176
Ibrahim Mammadov	21603109
Cenk Er	21600937
Nurlan Farzaliyev	21503756

Supervisor: İbrahim Körpeođlu

Jury Members: Hamdi Dibekliođlu and Özcan Öztürk

Final Report
May 27, 2020

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

Introduction	3
Requirements Details	3
Performance	3
Compatibility	4
Maintainability	4
Security	4
User interface	4
Scalability	4
Reliability	5
Final Architecture and Design Details	5
Cloud	6
Client	6
Development/Implementation Details	6
Client Application	6
Virtual Environment (Container)	7
Server and Storage	7
Database	8
Communication	9
Payment System	10
Distributed System	13
Testing Details	14
Maintenance Plan and Details	15
Other Project Elements	16
Consideration of Various Factors	16
Ethics and Professional Responsibilities	17
Judgements and Impacts to Various Contexts	17
Teamwork and Peer Contribution	18
Project Plan Observed and Objectives Met	22
New Knowledge Acquired and Learning Strategies Used	23
Conclusion and Future Work	25
More Secure VE	25
Distributed (clustering system)	25
SSH access to leaser	25
Payment	25
Notifications and mobile app	25
References	26

Final 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 using parts of those networks.

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]

2 Requirements Details

We had a number of different non-functional requirements specified in the Analysis Report. Requirements for most of them were met, however some of them were very hard to achieve and sustain.

2.1 Performance

We initially planned to implement our application in C++ for the sake of high performance, as we had virtualization as a major part of our application which was going to significantly drop our performance. C++ was considered as an option to compensate for the performance drop provided by virtualization tools. However due to great socket programming support and pyqt5 API (GUI library) provided by python we decided to use Python.

2.2 Compatibility

Rendt is a cross-platform application that supports a wide variety of systems and OSs. We achieved this by implementing Docker virtualization for actual code execution, on which you can run any language you wish. You can install Docker on Linux, MacOS, Windows (except Home edition which is still possible but more complicated) and this gives the opportunity to run Rendt on any major OS.

2.3 Maintainability

Rendt is utilizing a server to serve the users and renters. Server is responsible for payment, reliability ratings, rental requests and responses and displaying available PCs. Storage is responsible for file transfer. Server and Storage are maintained by us. Maintenance covers server rental payment, network issues with the server and providing the system online.

2.4 Security

Rendt is providing secure payment system. When the transaction starts time of execution is recorded and based on that time particular fee will be calculated. The renter can view and download his executed code only after the payment issues will be handled, and if leaser has some internet or electricity issues the operation will be removed, the renter will be notified about this and leaser's reliability points will be reduced. The execution environment is not safe completely because of one issue. Docker which we use as an virtual execution environment needs files to be on real hardware before it can take and use them in virtual environment. So for several milliseconds leaser will have chance to view send code before docker takes them, and also leaser will have several milliseconds to view output after docker will generate it and before it will be sent it to storage. On the other hand the execution process is completely safe because if there will be some malicious code send by renter it will crush docker environment but will have no effect on actual hardware.

2.5 User interface

UI is straightforward and a user without an advanced level of technical knowledge of computers will be able to operate the rendt app and be able to use it for renting or using rental PCs. Renters should have some technical background (especially in the distributed part) because they are in fact those who send code for execution. Leasers don't need to have any technical background, everything they need to know will be explained in the user manual (the installation and use of docker for example).

2.6 Scalability

System is able to serve as many requests at the same time. With the increasing number of requests and users, the system is able to serve without any compromises. Distributed execution part also allows renters to create clusters of any size.

2.7 Reliability

Rendt is a reliable service for users that need better performing hardware. From rendt's side, the server is up and running, serves the requests as necessary, provides security, scalability and compatibility. For the users' side, a user reliability rating system is implemented which is going to be formulated with the help of other users' feedback and reports which comes after operation abruption.

3 Final Architecture and Design Details

Four major components of our system are the central server, file storage site, database and client application. The server and file storage site are each deployed on an AWS EC2 instance. They both have access to our database instance, which is deployed on AWS RDS. The database instance only serves the server and storage site, but not the client application. The communication between server and storage instance is achieved through the database instance.

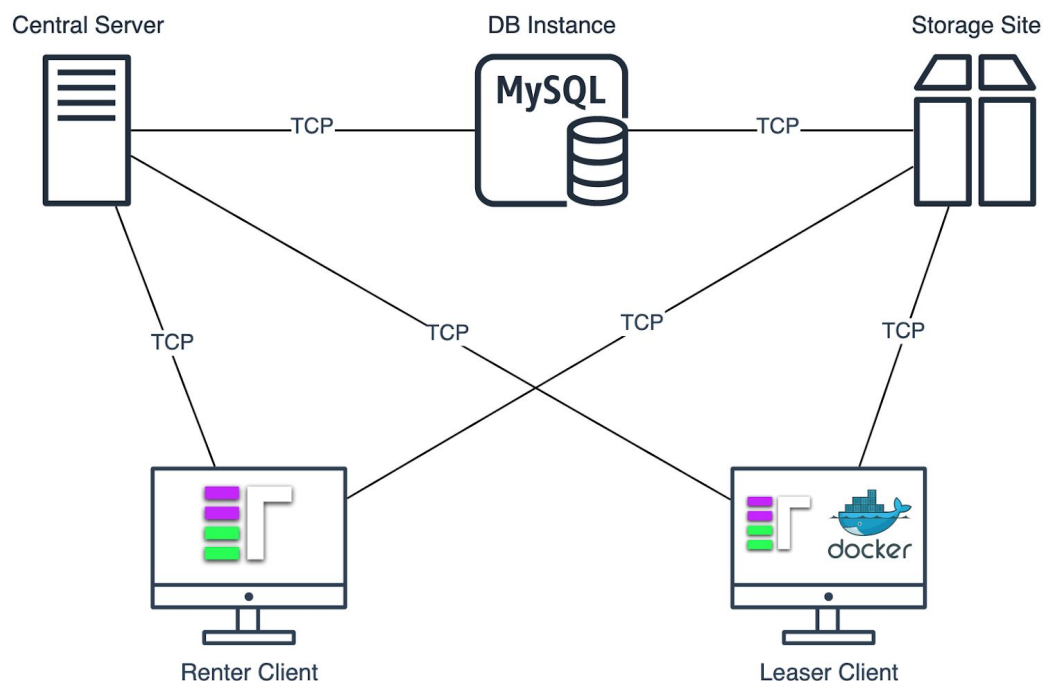


Figure 1: Rendt Architecture.

The client application stores the address of the central server and storage site, and sends/receives requests and files as necessary. In the case of a leaser (user that leases their computational resources), client application also creates and manipulates a Docker container for isolation and safe job execution.

We have a custom communication protocol and its implementation that runs on top of TCP and facilitates the communication between clients and server/storage. It defines

appropriate headers and standardizes message styles system-wide, and provides better message passing overall.

3.1 Cloud

We refer to the Central Server, DB instance and storage site collectively as the cloud. These form an abstraction of a unified server from the client's point of view. The server listens for TCP connections from the clients, internally processes their requests, queries/updates the database as needed, and answers clients back accordingly. It is responsible for authenticating users, granting upload/download permissions in the storage site, providing information about users' submitted job statuses etc.

The storage site is a simple version of the server instance; it receives client requests only regarding uploading/downloading files relating to jobs, check for permissions, and receives/provides the requested files. To use the storage site, i.e. to upload files pertaining to a job or download outputs of an executed job, the client app connects to the server and asks for a permission token. The server, having verified that the requested upload/download action is permissible and/or files are available, issues a storage token to the client. The server stores a copy of this token on the database instance. To perform the upload/download action, the client app sends the token it has previously received from the server to the storage site, where the actual files are stored. The storage site accesses the database and verifies user token, and upon success, starts and facilitates the file transfer action.

The database thus serves a double purpose; it facilitates the data storage of our entire ecosystem (users data, job transactions etc.), and also acts like a communication bridge between the server and the storage site.

3.2 Client

The client side of Rendt application consists of 2 major parts - sender and receiver (renter and leaser) objects. They inherit common properties from client class which is connected to client_messaging class (pipe between clients and server is created here). All other files related to GUI and authentication are connected with sender and receiver files, while docker related files are connected with only receiver (actual execution is triggered in receiver file). "Main function" is in the ui.py file from which you actually start the application.

4 Development/Implementation Details

4.1 Client Application

The GUI is developed with PyQt5. We used this library as it is one of the favorites for Python, but ultimately, Qt allows designing and deploying OS-independent applications for each OS, including mobile environments such as Android and iOS.

The final iteration of the design includes 2 Main Windows, namely, LoginWindow and LoggedInWindow. User first arrives at the Login window with all of its different views and widgets such as Login, Register and Forgot Password views. After logging into the system, LoggedInWindow and its components are loaded. This window contains 2 essential parts; Sidebar and Content. Sidebar contains SidebarElement instances and shows the user current page, available pages to go and lets the user to go through all the different stages of the application. Content is the dynamic part of the window and changes depending on the selected page.

Other than serving as a container, LoggedInWindow also has the necessary communication with the back-end, namely, receiver and sender classes. We may access and call back-end functions from this class. To do this, each view, page or class is initialized with a parent class. With a parent variable set, we may access these features and more by accessing these parent classes. By going up in the parent hierarchy we arrive at LoggedInWindow. Also this feature becomes handy for checking Docker information and status in various stages of the workflow.

4.2 Virtual Environment (Container)

We used docker containers to make the users code and leases computer less reachable and to create a common environment for all users, an environment that is the same for every renter and every leaser that runs on every operating system. Docker containers can run on every operating system and run any code that it's designed to run. Our containers run on an Ubuntu image. It has an unzip, Python, a default JDK, GCC compiler, Python3, pip, pip3 and renters can add further improvements by writing the bash code as they desire. Also, they can retrieve any trained code piece, any of the files they want to see after execution any output by simply adding print statements to their code if they want it returned to the bash code after execution. After making the decision of how we want to build the container image then we decided how to run. At first we were adding files for execution while building but then we decided it was a slow process and we should not need to build our Docker container image every time a leaser wants to lease their computer since they pay by the hour. So, then we made changes and we created a way to pass the zip into a running container, unzip it, then run the bash code inside it, take the output file and put it to the output directory where everything renter needs that she/he sent there zip that directory and send it back to the renter. So, that way if something bad is sent to someone it will be opened inside a container or if something goes wrong and it might damage files it will happen to the container and leasers will be safe.

4.3 Server and Storage

The serve and storage implementation are very similar; in fact, storage was built after the server using the same logic and code snippets, so we describe them together.

The server and storage are driven by multithreaded python applications that listen to client connections, and upon receipt thereof, serve their requests on a newly created thread. The server (and storage) accepts TCP connections and parses received data through the custom messaging protocol. Each client message has a field specifying the type of request that the client is making, e.g. sign-in. The server script has separate functionality for each request type, so according to the request type, a different code snippet is run.

The server and storage are each located on an AWS EC2 t2.micro instance. All throughout development, we connected to them via Secure Shell (ssh) and ran locally developed files and conducted tests. The server and storage code is implemented with extensive logging functionality, which makes it easy to debug and troubleshoot problems.

In view of the importance of user data privacy, the server and storage are configured to wrap all connections with TLS [5] (Transport Layer Security). Both have a self-signed certificate that can be verified by clients. All connections are thus encrypted and user information, in particular passwords and email addresses are safe.

To facilitate the communication with and data manipulation on the relational database, the server and storage are equipped with a custom database querying module (which uses python mysql-connector [6]), where all the necessary queries for the functionality of our system are implemented.

The server side also has a module containing cryptographic functions for hashing and salting user passwords, and for verifying them. Apart from these, the server is equipped with several token- and id-generators, which it uses for generating user IDs, job IDs, storage file tokens and others.

For file verification, files that are sent by storage to the clients are accompanied by a checksum [7] to validate the integrity of the files, and re-download if necessary.

4.4 Database

We have created our database instance on Amazon's RDS. We have chosen MySQL as the engine type for our database. We have preferred SQL over NoSQL since there are relations between the data types; stability and data integrity is needed, and we are not expecting changes or growth on our database etc. [8]. After creating the database we have used MySQL Workbench [9] as a tool to connect, manage, manipulate, modify etc. our database. It allows us to run the queries to make required changes or retrieve or check the data, and also due to its nice UI we could make some simple changes very fast, which saved us some time.

Tables in our database:

→ active_auth_tokens

- ◆ This table keeps track of the authentication tokens which are active at the moment by holding data of user_id, auth_token, time_issued
- archived_auth_tokens
 - ◆ This table keeps track of the authentication tokens which have been used before and being archived by holding data of user_id, auth_token, time_issued
- exec_file_tokens
 - ◆ This table stores tokens that give access to (executable) file upload to/download from the storage site. It holds job_id, db_token, file_size.
- output_file_tokens
 - ◆ This table stores tokens that give access to (output) file upload to/download from the storage site. It holds job_id, db_token, file_size.
- job_orders
 - ◆ This table keeps track of the jobs that have been submitted and accepted already by holding data of order_id, renter_id, job_id, job_desc, job_mode, file_size, leaser_id, status, exec_start_time, exexc_finish_time
- jobs
 - ◆ This table keeps track of the available jobs that have been submitted by the renters but not accepted by any leasers yet. The table hold data related to job_id, user_id, job_type, files_size, job_status, additional_comments)
- leasers
 - ◆ This table saves the data of the users who accept the jobs to be run on their computers by holding data of user_id, status, short_info, machine_details, price.
- passwords
 - ◆ This table keeps the passwords of the users by holding email_address and password_hash. Passwords are not stored in cleartext but are salted and hashed, for security reasons [10]. The reason we have saved the passwords in a different table is that it is considered to be safer.
- users
 - ◆ This table holds the data of users, such as user_id, email_address, username, user_type. User_type defines if the user is leaser or renter.

4.5 Communication

As mentioned in 4.3, all the nodes of our system except for database communicate with a custom messaging module. This module implements a protocol that must be followed for meaningful communication. For example, if the server is able to verify that an incoming message was sent from a valid Rendt application (according to the protocol), it proceeds to serve the request inside the message. To verify messages, the server application checks the headers and format of requests, and validates the completeness of data that accompanies them (for example, the server can't serve 'sign-in' request that hasn't provided an 'email-address').

The protocol we used is quite similar to HTTP. Each message contains a message body containing all useful information for server/storage to process the request, a message header describing the message body (encoding, size in bytes) and a mini-header that only comprises a 2 bytes integer describing message header size in bytes.

4.6 Payment System

After doing the research we realised that in our case it is almost impossible to create a payment system from scratch, since it required too much time. Therefore, we have decided to use or integrate already existing payment systems to our project. For implementing the payment system we are using Stripe API [11]. Stripe provides many kinds of payment methods. For our project we chose to implement a “Card Payment System”. The system has the client and server sides. After a payment assigned to the clients, they can proceed to the payment. We require clients to enter their card details, such as credit card number, expiration date, CVC code etc. in order to charge them. They can also enter their email addresses if they need to receive a receipt after the payment.

The security of the user data is the responsibility of the Stripe community [12], [13], [14]. They handle the privacy and safety of the card details and other sensitive information related to the users, such as addresses etc., during the transactions. In our system we do not keep the user details during the payment process, neither allow Stripe to save any sort of user data.

Transactions are handled by the Stripe as well. They send the required card details to the bank, get response and continue the process in accordance. The whole process is in the below Figure 2. The server is creating a PaymentIntent [15] then after handling the transaction it returns an intent object (as seen in the Figure 3) which gives us details about the payment process, payment status, id etc [16].

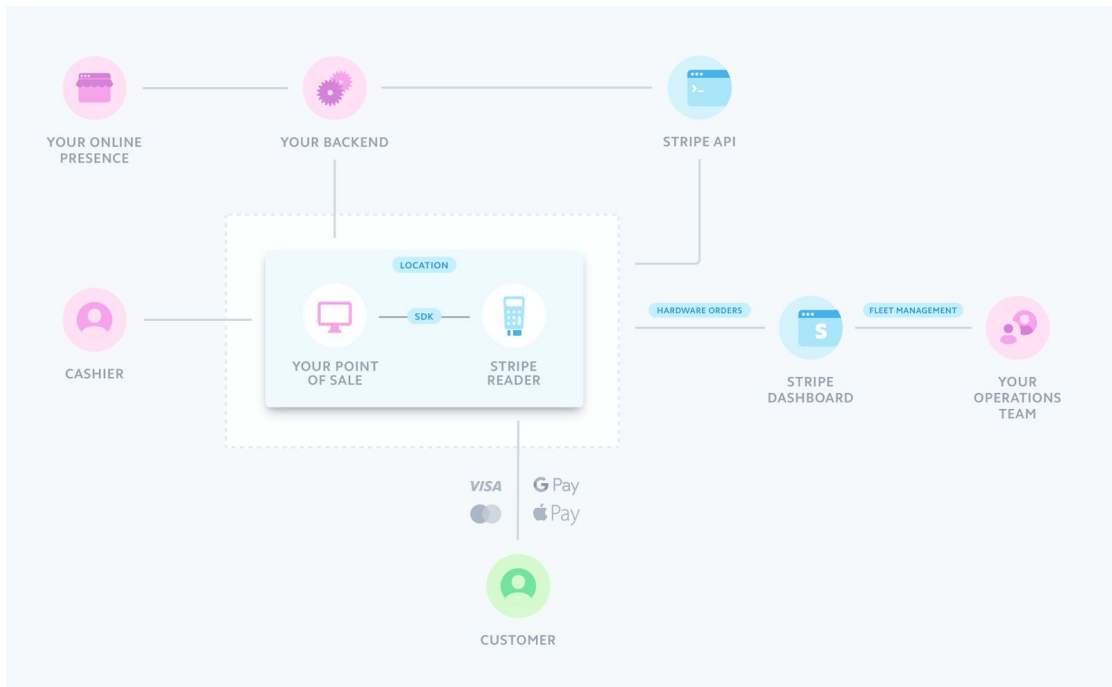


Figure 2: Stripe payment transaction cycle [17].

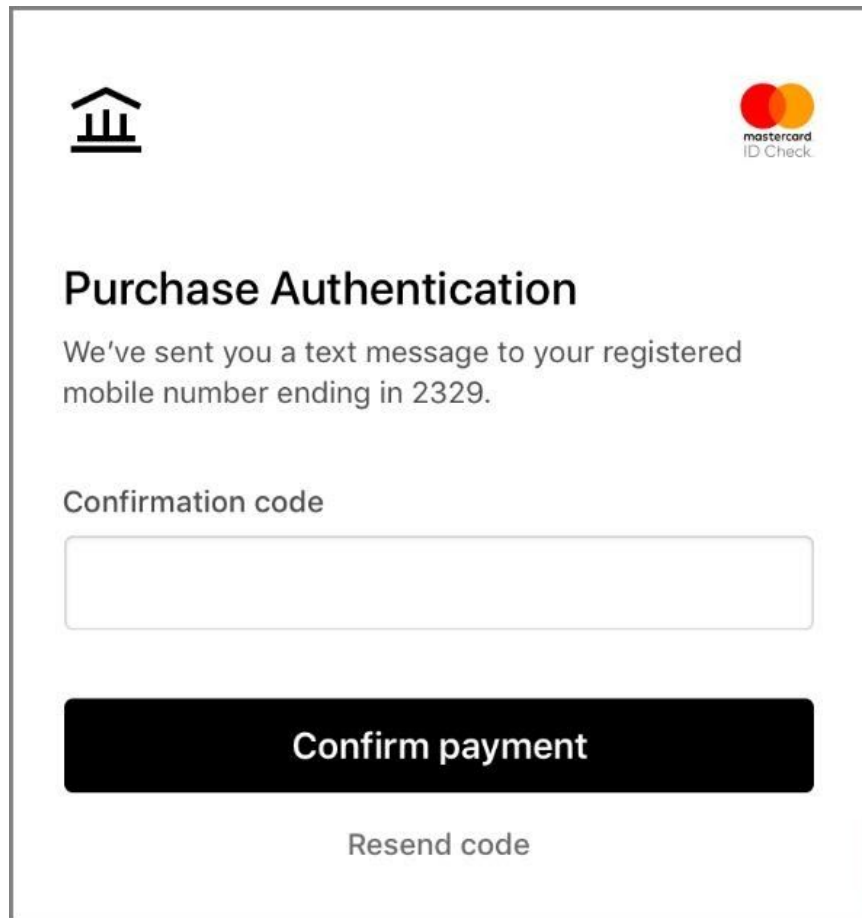
```
THE PAYMENTINTENT OBJECT

{
  "id": "pi_1Gln1VKFQLVwCuq086dEVqkJ",
  "object": "payment_intent",
  "amount": 1000,
  "amount_capturable": 0,
  "amount_received": 0,
  "application": null,
  "application_fee_amount": null,
  "canceled_at": null,
  "cancellation_reason": null,
  "capture_method": "automatic",
  "charges": {
    "object": "list",
    "data": [],
    "has_more": false,
    "url": "/v1/charges?payment_intent=pi_1Gln1VKFQLVwCuq086dEVqkJ"
  },
  "client_secret": "pi_1Gln1VKFQLVwCuq086dEVqkJ_secret_DP840tNqxfSjafCSvvdT",
  "confirmation_method": "automatic",
  "created": 1590200621,
  "currency": "usd",
  "customer": null,
  "description": null,
  "invoice": null,
  "last_payment_error": null,
  "livemode": false,
  "metadata": {},
  "next_action": null,
  "on_behalf_of": null,
  "payment_method": null,
  "payment_method_options": {
    "card": {
      "installments": null,
      "request_three_d_secure": "automatic"
    }
  },
  "payment_method_types": [
    "card"
  ],
  "receipt_email": "jenny.rosen@example.com",
  "review": null,
  "setup_future_usage": null,
  "shipping": null,
  "statement_descriptor": null,
  "statement_descriptor_suffix": null,
  "status": "requires_payment_method",
  "transfer_data": null,
  "transfer_group": null
}
```

Figure 3: Stripe payment intent object example [16].

We are handling possible errors during or before the transactions happen. Errors, such as invalid card number, expired cards, wrong CVC code etc. We also enabled 3D security protocol in our payment system. For example, if some banks require their

users to enter 3D security code, which is only a single-use, we provide a window [18] (see Figure 4) where users enter the security code they have received from their banks to proceed the payment.



The screenshot shows a mobile payment authentication screen. At the top left is a bank icon, and at the top right is the Mastercard ID Check logo. The main heading is 'Purchase Authentication'. Below it, a message states: 'We've sent you a text message to your registered mobile number ending in 2329.' There is a label 'Confirmation code' above a text input field. Below the input field is a large black button with the text 'Confirm payment'. At the bottom center, there is a link that says 'Resend code'.

Figure 4: 3D security requirement example [18].

4.7 Distributed System

In this part we faced big constraints and problems. We wanted to create a clustering system on our server where codes written for distributed systems could be partitioned and executed on different machines. We planned to test it with the MPI library for C. We created a specific page in our GUI for distributed execution. Renters were going to write distributed code, go to rendt and select distributed execution, after which they were going to choose a specific number of leasers they needed. First leaser was always server, so if number of chosen nodes is 4, number of actual leasers was going to be 3 (master node is always server). Then the code would be sent to the server, where the same instance of file would be created to accommodate all users(so if the number of leasers is 3, 4 instances would be created; 1 instance would run on the server). Server would distribute work to leasers by specifying their IPs and ports (port of the docker which will listen to the executed process) and execution of all instances would start at the same time(1 on servers, others on leaser's dockers). Everyone was going to execute same code ,but because IPs of all leasers are known

each leaser actually was going to execute the part of code assigned only to it (Renter should use argument variables instead of thread IDs and IPs was going to be assigned on nodes on the server; at the same time ports also should be specified as argument variables in the code and they would also be assigned appropriate values on server). This structure which was halfly implemented was considered to work until we faced a big challenge which we couldn't overcome. We were not able to find a port that we needed during node communications as it was not bounded with global IP, instead port which listened to process execution was bounded with local IP. This led to a situation where we weren't able to access the leaser's computers from other leasers or server (accessing server from users is possible because port forwarding is implemented automatically on all servers). So we came up with only 2 solutions to solve this problem. First is that all leasers are going to configure their routers for port forwarding. Second was to create our own custom pipes (in MPI because we were testing with it) which will work as our system for file transactions between users(client will try to connect to server which listens and waits for connection). First solution was limiting rendt usability, because it led to a situation where we required leasers to have technical background (renters should have technical background because they are those who write the code, but leasers shouldn't). Second solution was very hard to implement in a limited amount of time. So we decided to add distributed system implementation in our future plans.

5 Testing Details

The server side and basic client application (from which the current version has evolved) of our system were the first subsystems to be built, and throughout our developments they have been extensively tested and upgraded. It proved of crucial importance that we implemented logging functionality on both systems which allowed us to monitor and troubleshoot effectively.

The server and storage site are able to serve multiple clients simultaneously; they both have passed our tests, carried out by connecting several clients and monitoring the server, storage site and client responses. The storage site is capable of handling quite large files; we have tested it with jobs of size up to 2.2GB.

The client application has been tested on Windows 10 Education (Build 18363), macOS 10.15.2 and Ubuntu 18.04. GUI preserves its functionality and user-friendliness on all mentioned platforms. Given internet connection availability the application can connect to the server, submit a job and upload its files. Given, additionally, Docker availability, the application can receive executables from the server, oversee their execution in a docker container and send results back.

The client-server and client-storage communications have been tested for privacy using Wireshark. Secure communication was one of our requirements, and we have verified that all data traveling between the client and cloud is encrypted.

To test docker,we focused on running sample containers besides the main project without using certain things i.e. without receiving or sending files from/to a computer

and seeing the results and tested certain aspects what is capabilities can we extract files or manipulate them from our app and acted accordingly while merging it with the main app then see whether it worked exactly the same when we use it on the app.

Stripe can help us to accept payments from everywhere in the world. However, the company does not support merchant accounts for every country. What it is meant here is, you need to settle your business in one of the countries that Stripe supports merchants from. Unfortunately, Turkey is not in their list right now [19]. We have created an account to be able to accept the payments and send the money to the leasers. However, we could not activate the account and therefore we could not check the payments with the real cards. In order to activate we either had to benefit from Stripe Atlas [20] or get a merchant account in one of the countries in the list by the help of other third party companies, such as Wyoming Merchant Services. The problem with these options was that we had to pay a lot of money, around 1000 USD [20], [21], to those companies to get a business account. However, we did not pay that much money. Even if we did, we had to handle taxes as well, which was another problem for us to handle. On the one hand, Stripe allows us to test our payment system with the test cards they have provided [22]. We have checked our system for different payment brands, such as Visa, Visa (debit), Mastercard, Mastercard (debit) etc., we could get payments successfully done as it can be seen in Figure 5.

AMOUNT	DESCRIPTION	CUSTOMER	DATE
\$100.00 USD Succeeded ✓	pi_1GnCnbKFKQLVwCuq0waNAs4PM		May 27, 2020, 3:07 AM ...
\$100.00 USD Succeeded ✓	pi_1GnASYKFKQLVwCuq0Pw8QDoNG		May 27, 2020, 12:37 AM ...
\$100.00 USD Succeeded ✓	pi_1GnA0aKFKQLVwCuq0MUjQ2j5C		May 27, 2020, 12:33 AM ...
\$100.00 USD Succeeded ✓	pi_1Gn9kJKFKQLVwCuq0oyCDINTu		May 26, 2020, 11:51 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn9GvKFKQLVwCuq0quKmERTl		May 26, 2020, 11:21 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn9FJKFKQLVwCuq0eLwcU4Bj		May 26, 2020, 11:19 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8zCKFKQLVwCuq0hccHUsNB		May 26, 2020, 11:03 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8xdKFKQLVwCuq0unCMT8CY		May 26, 2020, 11:01 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8oIKFKQLVwCuq0GGIYyonj		May 26, 2020, 10:51 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8iVKFKQLVwCuq0i2yX37I6		May 26, 2020, 10:45 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8gMKFKQLVwCuq0fgLlYvVY		May 26, 2020, 10:43 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8cpKFKQLVwCuq0I6jQ89ww		May 26, 2020, 10:39 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8azKFKQLVwCuq0iZU2B0Tr		May 26, 2020, 10:37 PM ...
\$100.00 USD Succeeded ✓	pi_1Gn8ZJKFKQLVwCuq0E9j2aifz		May 26, 2020, 10:36 PM ...

Figure 5: Successful payments from Rendt’s account

6 Maintenance Plan and Details

For maintenance, we plan on keeping on with the updates of docker and further implement our system to keep them up to date of our users docker versions. We will make further monitoring on our instances in Amazon Web Services so that we will

stop using unnecessary resources or acquire more of them in terms of our needs. For example, we will test our servers how much request it can handle and scale it according to our needs or we can create an auto scaling server if it might get difficult to handle manually. We plan on making decisions by monitoring CPU and memory usage or how many requests it's handling. We will also maintain our storage units with the same procedures however for both cases since we are using free tier resources increasing our capacity will be according to our income and how much our application will be used. The main thing about storage maintenance is constant checking of logs, if something goes wrong it is possible (just like in server) to know what exactly happened by checking logs. 2 folders are created in the storage environment, one for codes sent by renter, other for output sent by leaser. By checking their existence and content we can verify if something went wrong during file transactions. Also we are planning on getting user feedback to listen to what they need, what is broken and what needs to be fixed immediately.

7 Other Project Elements

7.1 Consideration of Various Factors

Apart from all the factors above, we have taken into account some elements outside of the scope of the 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 the 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 are 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

7.2 Ethics and Professional Responsibilities

In view of all the factors in 7.1 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 sacrificed 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 with them based on this rate that anyone will see before submitting their task.

7.3 Judgements and Impacts to Various Contexts

→ Global Impact:

- ◆ There are no restrictions for any region in the world to use our system. So, everyone with powerful computers can benefit from the system as

a leaser and others can use the leaser's computers for their own purposes. And since payments can be accepted globally as well users will not need to get any sort of account from other countries just in order to use the system.

→ Economic Impact:

- ◆ Systems with similar purposes are already in the market. However, since it is being provided by the tech giants they can control the money flow in the business easily, and that is why benefitting from those systems are quite expensive. However, usage of our system will be much cheaper in comparison. If users use our system they do not have to pay a lot of money to buy new devices.

→ Societal Impact:

- ◆ One of our purposes was to help people in the society to make money with their own devices and others to do their jobs with the lowest possible charge.

Table 2. Impacts of Judgements

Judgement Description:	We decided to make Rendt a platform where everyday computer users with minimal technical knowledge can lease their computational resources in exchange for a fee.	
	Impact level	Impact Description
Impact in Global Context	8	Everyone can benefit from the system in two different ways: as a leaser or renter
Impact in Economic Context	10	Instead of paying lots of money to the different systems, by paying a lot less users can benefit from our system
Impact in Societal Context	7	Individuals can make money

7.4 Teamwork and Peer Contribution

Mahammad Shirinov:

1. Server

Built the initial server prototype with Huseyn for simple python file transfer between two machines, execution (in native OS) and output file transfer back to the first machine. Built the current server on top of this prototype;

implemented all client request types, their appropriate messages and corresponding code snippets to serve those requests on the server side. Implemented the responses to different requests, along with appropriate error messages if needed. Configured logging for maintenance and debugging purposes. Additionally, introduced client-server messaging protocol and implementation, multithreading for scalability and TLS encryption.

2. Authentication

Conducted initial research with Nurlan on authentication methods. Built an authentication service on client and server sides, which would communicate for user log in/sign up. Introduced salting and hashing functions on the server for safe password storage.

3. Client app backend

Built an initial client app with a command-line interface that could send files to the server, receive them, execute them, send back results and retrieve results. Later developed the client app back end on top of this implementation to include more functionalities and prepared requests that would be properly recognized and responded to by the server and storage.

4. Database queries

Wrote some database querying functions on the server side essential for server and storage functionality as necessary.

Ibrahim Mammadov:

1. GUI development:

Worked on the client-side front-end of the application and established connection between various front-end and back-end elements to provide functionality with design. Developed 3 iterations of the design depending on the changed requirements and developments in the project.

2. Fetching local Docker info:

A class for fetching local Docker daemon's configurations, resource usage, daemon existence and status.

3. Fetching machine details:

Getting machine details, such as CPU brand and model, number of logical processors, total memory of the system.

Huseyn Allahyarov:

1. Client side back-end logic implementation:

Particularly worked in client server connection establishment at the beginning of the project (was one of the first implemented parts of the project). In future participated in different updates (update of something that already was or add of new functionality) and bug fixes of client side back-end logic, especially during storage implementation.

2. Server logic implementation:

Participated mostly at the beginning stages when the server was created and some basic functionalities were added. In future occasionally participated in updates or bug fixes.

3. Storage implementation:

Implemented storage on his own (more details in the section 4).

4. Distributed system implementation:

Researched and implemented the distributed systems part, but unfortunately due to problems we faced (you can read about details in the implementation part) it was decided to postpone this part for future work.

Cenk Er:

1. Virtualization of execution environment:

Worked on making code execution and file access less reachable for leaser so that renters won't be afraid that their codes are only read by them or the outputs. Worked on making docker containers that can run python, java or C scripts with only a single bash file and users can download pip libraries too for their code or run any command for ubuntu to improve that container for their use for the time they need. Tried to make it as automated as possible to a certain degree and also we planned to create SSH between container and a renter so that their experience would improve and use everything more efficiently however we postponed that idea for the time being.

2. Notification system:

Worked on creating mail service for our users to get notifications about their job informations i.e. sending an email automatically when the leasers execution is done and output is uploaded to the server.

3. Distributed system implementation:

Got only the research part of the implementation to work on docker until it was postponed.

Nurlan Farzaliyev:

1. Database:

Created a database on Amazon's RDS. Then using MySQL Workbench created tables, made relations between them and by time when it is required did necessary modifications. Provided a class which helped to query and make changes to the database within the code, not using the tools.

2. Authentication:

Did research on authentication with Mahammad. Worked on providing a token that would be used to authenticate users. Also added encryption methods for using the data securely.

3. Payment System:

Did research on various payment systems on the market. After checking documentations has decided on which method to implement. Set up a card payment system for the project using Stripe API, which allows renters to make payments globally. Tested the implementation with the test cards provided by the Stripe.

7.5 Project Plan Observed and Objectives Met

Task Name	Duration	Start	Finish	Predecessors	% Complete	Resource Names
1.Group Formation	14 days	Mon 23.09.19	Sun 06.10.19		100%	All Group
2.Project Specification	14 days	Mon 07.10.19	Sun 20.10.19	1	100%	All Group
3.Analysis Report	28 days	Mon 21.10.19	Sun 17.11.19	2	100%	All Group
4.High-Level Design Report	44 days	Mon 18.11.19	Tue 31.12.19	3	100%	All Group
5.Research and Preparation for Project Implementation	33 days	Wed 01.01.20	Sun 02.02.20	4	100%	All Group
6.Low-Level Design Report	21 days	Mon 03.02.20	Sun 23.02.20	5	100%	All Group
7. GUI Implementation	94 days	Mon 24.02.20	Wed 27.05.20	5;6	90%	Ibrahim Mammadov
8.Client Side Back-End Logic Implementation	94 days	Mon 24.02.20	Wed 27.05.20	5;6	90%	Mahammad Shirinov,Huseyn Allahyarov
9.Sever Logic Implementation	94 days	Mon 24.02.20	Wed 27.05.20	5;6	100%	Mahammad Shirinov,Huseyn Allahyarov
10.Storage Implementation	31 days	Wed 01.04.20	Fri 01.05.20	5;6	100%	Huseyn Allahyarov
11.Database Implementation	31 days	Wed 01.04.20	Fri 01.05.20	5;6	100%	Nurlan Farzaliyev
12.Docker and Execution Enviroment Implementation	57 days	Wed 01.04.20	Wed 27.05.20	5;6	90%	Cenk Er
13. Authentication System Implementation	17 days	Wed 15.04.20	Fri 01.05.20	5;6	100%	Mahammad Shirinov,Nurlan Farzaliyev
14.Notification System Implementation	13 days	Fri 15.05.20	Wed 27.05.20	5;6	70%	Cenk Er,Mahammad Shirinov
Distributed System Implementation	18 days	Sun 10.05.20	Wed 27.05.20	5;6	20%	Huseyn Allahyarov,Cenk Er
Payment System Implementation	8 days	Wed 20.05.20	Wed 27.05.20	5;6	80%	Nurlan Farzaliyev

Figure 6: Project Plan

As you can observe from the table by the 27th of May we met almost all our objectives. Our actual work plan differs from the one we described in the analysis report in 2 major ways. The first difference is that it took more time than we planned for some parts of our project and the second is that we have no major dependencies in the implementation phase of our project. The reason for absence of dependencies is that everyone was working on their own part and only after partial completion of dependent parts was merged with the main branch of our project and if there were

still some independent work left; its implementation was continued. So for example when storage was completed by 50% and server by 60% they were merged because by that time all dependencies of storage from server were already met and the rest of the work in storage was only about it or concerned other parts of the project.

By 27th of May we almost finished our project. There are some small issues left in GUI, client logic, Docker (mostly security issues which are impossible to overcome) and payment system (some issues that must be handled, such as getting a merchant account and paying taxes). In the distributed system part we faced major problems which you can read about in detail in the implementation part (section 4).

7.6 New Knowledge Acquired and Learning Strategies Used

Learning strategies

We mostly focused on standard learning strategies as internet surfing and research, online courses, resources and documentations.

Knowledge Acquired

We learned a lot from this project. The first thing to mention is socket programming, which we applied to client-server and client-storage communications. We learnt to set up sockets, send/receive data and files over them and encrypt data transmitted over sockets. There turned out to be more nuances to data persistence across large systems, which we experienced when sending/receiving data over sockets in real time, having to store that data in the storage/database and (almost) simultaneously having to access that data from another endpoint/

For the GUI part, we learned a lot from the Qt [23] library's workflow and usage. The need for multithreading in application was not an expected requirement, but with trial and error, we implemented the multithreading to avoid bugs and freezing while using the app. Another crucial part is about the flexibility and responsiveness of the design. High-DPI scaling [24] can be a problematic part when developing an OS-independent application. This can cause problems such as non-smooth scaling of images, different size of fonts for the texts and labels used, given size details in pixels, unsupported fonts for different OSs, path details for different OS architectures and hierarchies etc. PyQt [23] library was another problematic part as a library. Firstly, the support for the library is different in each environment. Some OSs may not find or compile the library's components, some native features may require OS-dependent knowledge for seamless integration, multithreading can be problematic for Qt widgets as they require QThreads which are different than native threading libraries of the Python, deployment and packaging may cause problems for different OSs etc. Class hierarchy is another important aspect of PyQt as the differences between widgets

and windows can be unexpectedly crucial and may cause problems while running the application. Inheritance in widgets and parent-child communication sometimes is required for a better layout design and communication in-between different components. Qt C++ and PyQt may differ as well as different versions of the PyQt. Documentation is mostly up-to-date, however, some library differences are not highlighted. Qt is also supported by 2 official libraries, namely, PySide [25] and PyQt. Differences between these 2 libraries are also another factor to be considered when choosing the right library for the development. Design details, colors, alpha values, drop-shadow effects, font-weights, transparency and other factors play a huge role in the development. If these factors are not properly specified, the necessary features cannot be implemented or provided. Cleanup of the instances, hiding and destruction of various components while surfing through different views can cause problems for the user as well as the developer.

For the Docker part, we learned important syntax to build, create and run and docker container images. We improved ourselves by acquiring more knowledge about container networking, container volumes, how containers ports work. Also, we learned how much accessible images we have on public repositories on Dockerhub and the possibilities of what can be done with containers, and how it helps various processes.

To be able to set up payment systems we have done a lot of market research. We had some options that could be considered for global transactions. Some of them are PayPal, Stripe, Maxipago etc. Making payments with PayPal and some other systems is not supported by some countries. After doing some research we have concluded that Stripe is our best option, not only that it supports payments globally but also it had the best documentation that helped us to set up a payment system to our project. After setting up the payment system we have learned how the real transactions work, what is required to make global transactions and so on. Additionally, in order to run and check our payment system locally we have learned and used Flask.

For distributed job execution, even though we were not able to solve the problem we faced, we acquired significant knowledge about the potential issues and fundamentals of MPI. We clearly understood what port forwarding means and realized the problems it posed to distributed computation on personal computers (as opposed to servers), how IPs and ports could be used in MPI code instead of thread IDs, learned a lot about MPI environment variables (communication variables; default one is `MPI_COMM_WORLD` for example), how to assign specific ports to listen to MPI execution and how to analyze contents of ports. Also worth to mention that we learned a lot about different MPI functions created specifically for real distributed environments (`connect`, `disconnect`, `accept` functions etc.). We researched a lot about clustering systems to come up with an implementation plan we described in the implementation part (section 4).

8 Conclusion and Future Work

8.1 More Secure VE

We plan on creating a more secure, more reliable virtual environment than docker can offer so we are planning on changing our container system. Also if nothing we find helps us in the way we need, we might create our own container/virtual environment system and use it to create more secure environments for our users.

8.2 Distributed (clustering system)

Future plans include solving/circumventing port forwarding problems. Until now we came up with 2 solutions, but there could be more. First one (not preferred, but the easier one) is to divide leasers in 2 groups : port forwarded ones and not port forwarded and allowing only port forwarded leasers to participate in distributed execution. Second solution (preferred, but harder) is to create custom pipes between leasers and our server where the server is going to listen to leasers' requests, but this solution has its own bottleneck. The problem is that in the second solution communication between different nodes is only possible through the server (master node). This means that leasers (workers) will not be able to communicate directly with each other, they always should use a server (master) as a controller who redirects their requests to other leasers, but this solution doesn't require the division of leasers in two parts. So both solutions have advantages and disadvantages, and they will be considered in depth for future work.

8.3 SSH access to leaser

We would like to create an SSH tunnel between the renter's computer and the container instance on the leaser's computer to provide a more advanced experience so that renters can accomplish more things with this application and create a more functional environment for both users.

8.4 Payment

At the moment we can successfully make payments. However, as mentioned before since our Stripe account is activated we cannot send the money to bank accounts from the Stripe account. We would like to get a business account in the future and connect that merchant account with our Stripe account to be able to benefit from other functionalities of the Stripe as well.

8.5 Notifications and mobile app

For the future, we are planning to develop an app for users for getting notifications while away from their machines, accepting/rejecting requests, leasing and un-leasing their machines, paying for the execution in the app, seeing resource usage, request

details and statuses of their tasks, uptime, and more. With the addition of such an app, we may achieve our ultimate goal of full automation and minimal interaction between leaser and the machine as well as work efficiency on the go.

9 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] Vice. 2020. *Seven Ways To Donate Your Computer's Unused Processing Power*. [online] Available at: [<https://www.vice.com/en_us/article/bmj9jv/7-ways-to-donate-your-computers-unused-processing-power>](https://www.vice.com/en_us/article/bmj9jv/7-ways-to-donate-your-computers-unused-processing-power) [Accessed 27 May 2020].
- [5] 2020. [online] Available at: [<https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/>](https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/) [Accessed 27 May 2020].
- [6] Dev.mysql.com. 2020. *Mysql :: Mysql Connector/Python Developer Guide*. [online] Available at: [<https://dev.mysql.com/doc/connector-python/en/>](https://dev.mysql.com/doc/connector-python/en/) [Accessed 27 May 2020].
- [7] En.wikipedia.org. 2020. *File Verification*. [online] Available at: [<https://en.wikipedia.org/wiki/File_verification>](https://en.wikipedia.org/wiki/File_verification) [Accessed 27 May 2020].
- [8] Chan, M., 2020. *SQL Vs. Nosql - What's The Best Option For Your Database Needs? - Thorn Technologies*. [online] Thorn Technologies. Available at: [<https://www.thorntech.com/2019/03/sql-vs-nosql/>](https://www.thorntech.com/2019/03/sql-vs-nosql/) [Accessed 27 May 2020].
- [9] Mysql.com. 2020. *Mysql :: Mysql Workbench*. [online] Available at: [<https://www.mysql.com/products/workbench/>](https://www.mysql.com/products/workbench/) [Accessed 27 May 2020].
- [10] Crackstation.net. 2020. *Secure Salted Password Hashing - How To Do It Properly*. [online] Available at: [<https://crackstation.net/hashing-security.htm>](https://crackstation.net/hashing-security.htm) [Accessed 27 May 2020].
- [11] Stripe.com. 2020. *Stripe API Reference*. [online] Available at: [<https://stripe.com/docs/api>](https://stripe.com/docs/api) [Accessed 27 May 2020].
- [12] Stripe.com. 2020. *Services Agreement - United Kingdom | Stripe*. [online] Available at: [<https://stripe.com/ssa>](https://stripe.com/ssa) [Accessed 27 May 2020].
- [13] Stripe.com. 2020. *Privacy Policy - United Kingdom | Stripe*. [online] Available at: [<https://stripe.com/privacy>](https://stripe.com/privacy) [Accessed 27 May 2020].

- [14] Stripe.com. 2020. *Integration Security Guide* | *Stripe*. [online] Available at: <<https://stripe.com/docs/security>> [Accessed 27 May 2020].
- [15] Stripe.com. 2020. *The Payment Intents API* | *Stripe Payments*. [online] Available at: <<https://stripe.com/docs/payments/payment-intents>> [Accessed 27 May 2020].
- [16] Stripe.com. 2020. *Stripe API Reference - Paymentintents*. [online] Available at: <https://stripe.com/docs/api/payment_intents> [Accessed 27 May 2020].
- [17] Stripe.com. 2020. *Stripe Terminal Overview* | *Stripe Terminal*. [online] Available at: <<https://stripe.com/docs/terminal>> [Accessed 27 May 2020].
- [18] Stripe.com. 2020. *3D Secure Card Payments* | *Stripe Payments*. [online] Available at: <<https://stripe.com/docs/payments/3d-secure#web>> [Accessed 27 May 2020].
- [19] Stripe.com. 2020. *Stripe: Countries - Find Stripe In Your Country*. [online] Available at: <<https://stripe.com/global>> [Accessed 27 May 2020].
- [20] Stripe.com. 2020. *Stripe Atlas: Turn Your Idea Into A Startup*. [online] Available at: <<https://stripe.com/atlas>> [Accessed 27 May 2020].
- [21] Services, W., 2020. *Wyoming Merchant Services*. [online] Valuedmerchants.com. Available at: <<https://www.valuedmerchants.com/wyoming-merchant-services>> [Accessed 27 May 2020].
- [22] Stripe.com. 2020. *Testing* | *Stripe Payments*. [online] Available at: <<https://stripe.com/docs/testing>> [Accessed 27 May 2020].
- [23] Riverbankcomputing.com. 2020. *Pyqt5 Reference Guide — PyQt V5.14.1 Reference Guide*. [online] Available at: <<https://www.riverbankcomputing.com/static/Docs/PyQt5/>> [Accessed 27 May 2020].
- [24] En.wikipedia.org. 2020. *Resolution Independence*. [online] Available at: <https://en.wikipedia.org/wiki/Resolution_independence> [Accessed 27 May 2020].
- [25] Doc.qt.io. 2020. *Qt For Python Documentation — Qt For Python*. [online] Available at: <<https://doc.qt.io/qtforpython/contents.html>> [Accessed 27 May 2020].