
Bilkent University

Department of Computer Engineering

# Senior Design Project

*Project short-name: rendt*

## Low-Level Design Report

Project Group Members:

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

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

Low-Level Design Report
February 17, 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**

Object design trade-offs 3

Usability vs. Functionality 3

Native vs. Virtual 3

Portability vs. Security 4

Interface documentation guidelines 4

Engineering standards (e.g., UML and IEEE) 4

Definitions, acronyms, and abbreviations 4

**Packages** **6**

Server 7

Logic Tier 7

Data Tier 8

Client 8

Controller 8

Presentation 9

**Class Interfaces** **10**

Server 10

Logic Tier 10

Data Tier 14

Client 14

Controller 14

Presentation 18

**References** **23**

# 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 Object design trade-offs

### 1.1.1 Usability vs. Functionality

Rendt aims to appeal to people from every background, for the leasing part at least. To avoid any dependence on technical background, we aim to design our user interface adequately straightforward for ease of use and minimal user interaction for the leasers. For the renters however, technical background is an important factor. Nevertheless, UI will be easy to use for both types of users. By this choice, we may trade advanced functionalities of the system for ease of use and larger user base.

### 1.1.2 Native vs. Virtual

To provide the necessary security, we are aiming for a virtual environment instead of native execution. However, latest developments in CPU technology now let users utilize Hyper-V or similar technologies to offer native-like performance on virtual environments. Our aim is to minimize hardware damage and application misuse by providing a Docker container for the executions. By using containers, we will also be able to follow the same execution style for every user and computer.

### 1.1.3 Portability vs. Security

Rendt requires Docker installation for leasing. Docker installation may increase the size of the application, but with Docker installation we are aiming to provide a fully secure execution environment. Any type of misuse or hardware damage will not be reflected on the real hardware or OS components of the system, instead on the containers and related parts.

## 1.2 Interface documentation guidelines

| **Class** | ClassName |
|---|---|
| Class description | |
| **Attributes** | |
| attribute | attribute description |
| **Methods** | |
| method | method description |

## 1.3 Engineering standards (e.g., UML and IEEE)

In this report, for our models such as class, package, object etc. diagrams we followed UML guidelines. Since in our classes and almost all diagrams we used UML guidelines we considered this would be more appropriate. We followed the IEEE citation style for our references because this citation method is common among engineers.

## 1.4 Definitions, acronyms, and abbreviations

**RENDT**: Remote Execution and Distribution of Tasks

**Renter**: User looking for a leased computer to be rented

**Leaser**: User leasing his/her computer for rental

**Listing**: Collection of data concerning a leaser's leasing terms and conditions (performance, availability etc.)

**Remote**: Distant as in computer(s) not nearby

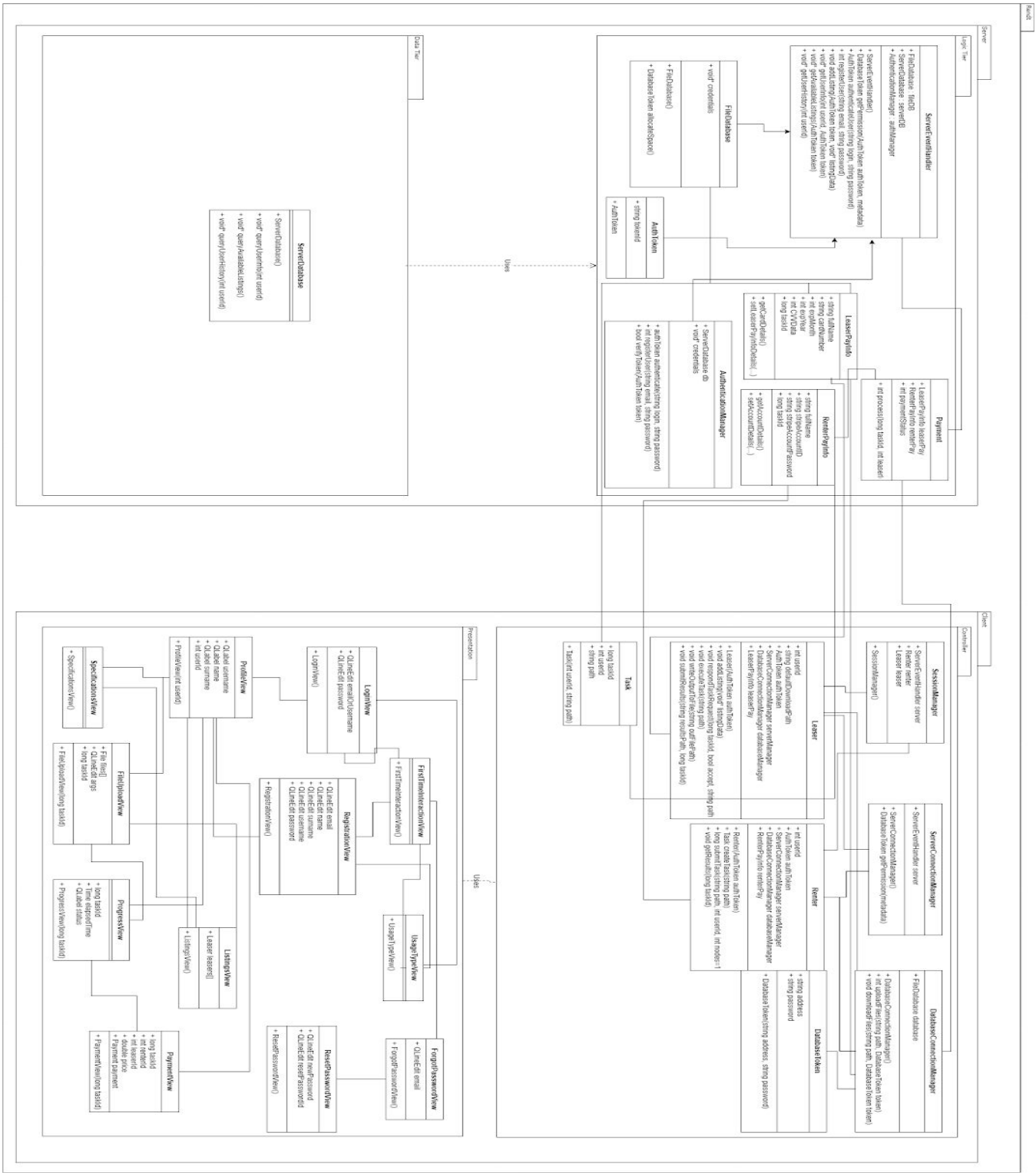
**Execution**: Compiling and running a task to get a result

**Distribution**: Running different tasks of the program on multiple leased computers for performance

**Task**: Unit(s) of program to be run and executed

| | |
|---|---|
| **Virtualization**: | A virtual environment for tasks to be executed safely |
| **VM**: | Virtual Machine (or Environment) |
| **OS**: | Operating System |
| **Cross-platform**: | Compatible with multiple platforms and devices |
| **UML:** | Unified Modeling Language |
| **IEEE:** | Institute of Electrical and Electronics Engineers |

# 2 Packages


Root
Server
Logic Tier
ServerEventHandler
+ FileDatabase fileDB
+ ServerDatabase serverDB
+ AuthenticationManager authManager
+ ServerEventHandler()
+ DatabaseToken getPermission(AuthToken authToken, metadata)
+ AuthToken authenticateUser(string login, string password)
+ int registerUser(string email, string password)
+ void addListing(AuthToken token, vector listingData)
+ void* getUserInfo(int userId, AuthToken token)
+ void* getAvailableListings(AuthToken token)
+ void* getUserHistory(int userId)
FileDatabase
+ void* credentials
+ FileDatabase()
+ DatabaseToken allocateSpace()
AuthToken
+ string tokenId
+ AuthToken
LeasePayInfo
+ string fullName
+ string cardNumber
+ int expMonth
+ int CVV
+ long taskId
+ getCardDetails()
+ setLeasePayInfoDetails( )
Payment
+ LeasePayInfo leasePay
+ RenterPayInfo renterPay
+ int paymentStatus
+ int process(long taskId, int leaseI
RenterPayInfo
+ string fullName
+ string stripeAccountID
+ string stripeAccountPassword
+ long taskId
+ getAccountDetails()
+ setAccountDetails( )
AuthenticationManager
+ ServerDatabase db
+ void* credentials
+ AuthToken authenticate(string login, string password)
+ int registerUser(string email, string password)
+ bool verifyToken(AuthToken token)
Data Tier
ServerDatabase
+ ServerDatabase()
+ void* queryUsers(int userId)
+ void* queryAvailableListings()
+ void* queryUserHistory(int userId)
Uses
Client
Controller
SessionManager
+ ServerEventHandler server
+ Renter renter
+ Lease lease
+ SessionManager()
ServerConnectionManager
+ ServerEventHandler server
+ ServerConnectionManager()
+ DatabaseToken getPermission(metadata)
DatabaseConnectionManager
+ FileDatabase database
+ DatabaseConnectionManager()
+ int uploadFile(string path, DatabaseToken token)
+ void downloadFile(string path, DatabaseToken token)
Lease
+ int userId
+ AuthToken authToken
+ string defaultDownloadPath
+ ServerConnectionManager serverManager
+ DatabaseConnectionManager databaseManager
+ LeasePayInfo leasePay
+ Lease(AuthToken authToken)
+ void addListing(vector listingData)
+ void respondToInfoRequest(long taskId, bool accept, string path
+ void executeTask(string path)
+ void writeOutputToFile(string outFilePath)
+ void submitResults(string resultsPath, long taskId)
Renter
+ int userId
+ AuthToken authToken
+ ServerConnectionManager serverManager
+ DatabaseConnectionManager databaseManager
+ RenterPayInfo renterPay
+ Renter(AuthToken authToken)
+ Task createTask(string path)
+ long submitTask(string path)
+ void getResults(long taskId)
DatabaseToken
+ string address
+ string password
+ DatabaseToken(string address, string password)
Task
+ long taskId
+ int userId
+ string path
+ Task(int userId, string path)
Uses
Presentation
LoginView
+ QLineEdit emailOrUsername
+ QLineEdit password
+ LoginView()
FirstTimeInteractionView
+ FirstTimeInteractionView()
RegistrationView
+ QLineEdit email
+ QLineEdit display
+ QLineEdit username
+ QLineEdit password
+ RegistrationView()
ForgotPasswordView
+ QLineEdit email
+ ForgotPasswordView()
ResetPasswordView
+ QLineEdit newPassword
+ QLineEdit resetPasswordId
+ ResetPasswordView()
ProfileView
+ QLabel username
+ QLabel name
+ QLabel surname
+ int userId
+ ProfileView(int userId)
UsageTypeView
+ UsageTypeView()
ListingsView
+ Lease leases[]
+ ListingsView()
SpecificationsView
+ SpecificationsView()
FileUploadView
+ File files[]
+ QLineEdit args
+ long taskId
+ FileUploadView(long taskId)
ProgressView
+ long taskId
+ Time elapsedTime
+ QLabel status
+ ProgressView(long taskId)
PaymentView
+ long taskId
+ int renterId
+ int leaseId
+ double price
+ Payment payment
+ PaymentView(long taskId)

## 2.1 Server

Server is the component in rendt's architecture that is responsible for overseeing job allocation and delivery of results, as well as payment. It is organized into a Logic Tier and Data Tier.

### 2.1.1 Logic Tier

The logic tier incorporates all the vital responsibilities of the server. It authenticates users, handles requests that come to the server and oversees task execution and results delivery between renter and leaser nodes. It also provides payment services.

**AuthenticationManager:** is responsible for authentication of users: creating accounts and accessing existing ones. When a client logs in to their account, they are issued an AuthToken that the client then uses in all their requests to the server.

**ServerEventHandler:** is the main processing object in the server. All the communication between client and server (including login and signup) passes through this point. A renter will communicate with this object before sending a task, and ask for permission. ServerEventHandler object will issue permission and will allocate (using FileDatabaseManager object, see below) space on the File Database for the files to be uploaded to, and will share a token that will allow this upload with the client. Also, after a leaser has executed a task, it will again ask for permission to upload results, and this object will issue permission and database token for uploading of results. Other than that, event handler object will receive data queries from the client (available listings, profile info), query it from Server Database and serve that information to the Client.

**FileDatabaseManager:** is responsible for overseeing the data transfer between renters and leasers through the separate database component. It will allocate space as needed, and issue tokens for secure access to this space.

**DatabaseToken:** a token object that will contain address information and secure key to access (upload to/download from) File Database.

**AuthToken:** a token object issued to a user when logging in which will authorize all server requests.

**LeaserPayInfo:** This class represents a leaser payment info. Its objects hold respective leaser's card details such as the owner's name, card number, expiration date and so on.

**RenterPayInfo:** This class represents a renter payment info. Its objects hold respective account details that belong to the respective user.

**Payment:** This class represents a payment object. It will handle the money transactions, basically the payments at the end of the sessions.

### 2.1.2 Data Tier

This layer is responsible for storing data about the user profiles, active listings and also past actions that have taken place on the server. It also contains private login information.

**Server Database:** is an object for accessing the database internal to the server, which is used to store

- login data
- user profile data
- current and past listings
- past transactions between users and some relevant information about them, like duration, cost, outcome, reliability etc.

## 2.2 Client

Client subsystem consists of presentation (view) and controller. These are respectively the GUI and functionality managers of client side subsystem.

In controler part there are a bunch of classes that maintain the local side of the application by identifying different types of users(leaser and renter), introducing connection managers for server and database, creating tasks and defining ultimate manager for the whole session of the application. Most parts of the client controller, especially task execution in leaser object, is executed in a virtual environment. In fact, the virtual environment is responsible for running jobs on leaser clients that are sent by senders. We will use Docker containers to run/execute the files that are sent by the renters to create virtual environment for the file and to prevent leasers to access those files.

### 2.2.1 Controller

This controller package is for the main functionalities of the application.

**ServerConnectionManager:** This class will manage connection between the server and the clients. Mainly it will ask for connection permission with the server so that the files of renter could be uploaded to the database. Database token will be needed for getting permission from the server

**DatabaseConnectionManager:** This class will manage connection with the database that will be used during sending and receiving of execution files. After server connection manager receives permission and token database connection manager uploads or downloads files from the database depending on the operation

**SessionManager:** This class is the main one in the whole system. First it logs in the user and obtains authentication token and establishes the connection with the server. Then depending on users demand and authentication token it log in either leaser or renter account.

**Renter:** This class represents a renter object. It establishes the connection with the server, creates tasks from provided files and their

locations and calls SeverConncetionManager and DatabaseConnectionManager in specified order for the further flow of the sending process. After the task is executed the same procedure should be in order to download executed files and the payment system should be called at the very end.

**Leaser:** This class represents a leaser object. Technically it is very similar to Renter. This object communicates with the server, downloads files for a specific task, executes them and uploads back the results.

**Task:** This class represents a task object. By using user id and file locations in the local system it creates an object of the task and assigns it a unique task id the is going to be referred to throw out all transaction.

### 2.2.2 Presentation

This presentation package is for the user interface of the application.

**LoginView:** This view is for logging into the system. By using username/email and password, users will be able to sign in and use the application as a renter or leaser.

**ForgotPasswordView:** This view is for requesting a password reset in case of forgetting the password. Users need to enter the email address of the account to send a password reset request. A custom ID, valid for an hour, will be generated and sent to the user for the request to be finished and resetting the password.

**ResetPasswordView:** This view will follow the ForgotPasswordView screen and with custom ID and new password entered, users will be able to reset their password and sign into the system.

**RegistrationView:** This view is for the users to create an account and start to use the application. With the entered credentials a new account will be created for the user with a uniquely generated ID.

**ProfileView:** This view will display the profile information of a user. If shown profile is the users' own, they will be able to edit their information and credentials on the same screen. Important profile credentials such as email address will be accessible only by the profile owners.

**FirstTimeInteractionView:** This view will follow logging into the system and will show quick tips and instructions for using the application once the application is started for the first time. In the latter executions, this view will not be shown.

**UsageTypeView:** This view will be presented to the user for selection of usage types, namely, renter and leaser. Once the user selects renter or leaser type, he/she will be directed to the following view.

**ListingsView:** This view will be shown to the renters and will list the current leasers and available leased computers. Adequate information with

important specifications of the system will be shown to the user for selection of the system/computer.

**SpecificationsView:** This view will be shown to the leasers and will allow them to specify the hardware configuration and availability of their system for leasing. Leasers will also select the availability duration for their computers and price for the leasing.

**FileUploadView:** This view will be shown to the renters who have successfully rented a leased computer to upload their files to be executed. After selecting the files to be executed, files will be uploaded to the leased computer and upload progress will be shown to the renters.

**ProgressView:** This view will display the status and progress of the executions to the renter with session's elapsed time. Ideally, outputs from the files will be shown directly to the renters. Renters will be able to download the resulting files from the leased computer.

**PaymentView:** This view will be shown to the renter after the session ends. With the selected payment method, renters will pay for the execution duration and end the rental session.

# 3 Class Interfaces

## 3.1 Server

### 3.1.1 Logic Tier

| **Class** | ServerEventHandler |
|---|---|
| Object that handles server events | |
| **Attributes** | |
| FileDatabase fileDB | instance of File database object |
| ServerDatabase serverDB | instance of Server database object |
| AuthenticationManager authManager | instance of AuthenticationManager object |
| **Methods** | |
| ServerEventHandler() | class constructor |
| DatabaseToken getPermission(AuthToken authToken, metadata) | asks for permission to upload files to the File database |
| AuthToken authenticateUser(string login, string password) | communicates with authentication manager and returns an authentication token to user |

| | |
|---|---|
| int registerUser(string email, string password) | communicates with authentication manager and creates a new user account, returns user ID |
| void addListing(AuthToken token, void* listingData) | adds a listing (sent from a leaser) to the database of listings |
| void* getUserInfo(int userId, AuthToken token) | queries user information from Server Database and returns (to the client) |
| void* getAvailableListings(AuthToken token) | queries available listings from Server Database and returns (to the client) |
| void* getUserHistory(int userId) | queries the Server database for past transactions of a user and returns (to the client) |

| **Class** | AuthenticationManager |
|---|---|
| Provides authentication for users. | |
| **Attributes** | |
| ServerDatabase db | instance of server database |
| void* credentials | credentials for accessing the server database |
| **Methods** | |
| authToken authenticate(string login, string password) | authenticates user and returns token for communication with the server and database |
| int registerUser(string email, string password) | creates a new user account, returns user ID |
| bool verifyToken(AuthToken token) | checks if a given token is valid |

| **Class** | FileDatabase |
|---|---|
| Object that handles temporary transfer of data between users | |

| **Attributes** | |
|---|---|
| void* credentials | credentials for accessing the file database |
| **Methods** | |
| FileDatabase() | class constructor |
| DatabaseToken allocateSpace() | allocates space for files to be uploaded and returns token to allow upload/download |

| **Class** | AuthToken |
|---|---|
| Token file that gives access to database upload/download actions. | |
| **Attributes** | |
| string tokenId | token ID |
| **Methods** | |
| AuthToken() | class constructor. generates token |

| **Class** | LeaserPayInfo |
|---|---|
| Object that holds leaser’s payment information/card details | |
| **Attributes** | |
| long taskId | ID of the task |
| string fullName | name of the card owner |
| string cardNumber | card number |
| int expMonth | expiration month |
| int expYear | expiration year |
| int CVVData | card CVV data |
| **Methods** | |
| getCardDetails() | returns a list of card details |

| | |
|---|---|
| setLeaserPayInfoDetails(...) | sets leaser's card details, such as name, card number, expiration month/year |

| **Class** | RenterPayInfo |
|---|---|
| Object that holds renter's payment information/account details | |
| **Attributes** | |
| long taskId | ID of the task |
| string fullName | name of the account owner |
| string stripeAccountId | ID of the stripe account of the renter |
| string stripeAccountPassword | password of the stripe account |
| **Methods** | |
| getAccountDetails() | returns a list of account details |
| setAccountDetails(...) | sets renter's account details such as name, stripe account ID/password with the given arguments |

| **Class** | Payment |
|---|---|
| Object that deals with the transactions | |
| **Attributes** | |
| LeaserPayInfo leaserPay | LeaserPayInfo object |
| RenterPayInfo renterPay | RenterPayInfo object |
| int paymentStatus | status of payment |
| **Methods** | |
| int process(long taskId, int leaserId) | finishes the transaction by getting transaction and account details of the partakers and returns payment status |

### 3.1.2 Data Tier

| **Class** | ServerDatabase |
|---|---|
| Object that manages data about listings, users and history | |
| **Attributes** | |
| | |
| **Methods** | |
| ServerDatabase() | class constructor |
| void* queryUserInfo(int userId) | queries the SQL database for info of the specified user |
| void* queryAvailableListings() | queries the SQL database for all active listings |
| void* queryUserHistory(int userId) | queries the SQL database for past transactions of a user |

## 3.2 Client

### 3.2.1 Controller

| **Class** | SessionManager |
|---|---|
| Central object on the Client side that is run when initiating the program | |
| **Attributes** | |
| ServerEventHandler server | event handler instance of the server. logs in the user and obtains token |
| Renter renter | renter object corresponding to this client machine |
| Leaser leaser | leaser object corresponding to this client machine |
| **Methods** | |
| SessionManager() | creates (gets) the Server Event Handler object from the server. initializes the client program |

| **Class** | Renter |
|---|---|
| Object of renter client | |
| **Attributes** | |
| int userId | id of the user |
| AuthToken authToken | authentication token of user |
| ServerConnectionManager serverManager | server connection manager object |
| DatabaseConnectionManager databaseManager | database connection manager object |
| RenterPayInfo renterPay | object to provide renter's bank account details for transactions |
| **Methods** | |
| Renter(AuthToken authToken) | class constructor. creates Renter object and establishes connection with the server and database. |
| Task createTask(string path) | creates a task object |
| long submitTask(string path, int userId, int nodes=1) | creates and submits created task to the server. returns taskId. |
| void getResults(long taskId) | download the results of an executed task |

| **Class** | Leaser |
|---|---|
| Object of leaser client | |
| **Attributes** | |
| int userId | id of the user |
| string defaultDownloadPath | location on client's machine where to download task files |
| AuthToken authToken | authentication token of user |
| ServerConnectionManager serverManager | server connection manager object |

| | |
|---|---|
| DatabaseConnectionManager databaseManager | database connection manager object |
| LeaserPayInfo leaserPay | user's payment details for transactions and payments |
| **Methods** | |
| Leaser(AuthToken authToken) | class constructor. creates Renter object and establishes connection with the server and database. |
| void addListing(void* listingData) | submits a listing to the Server |
| void respondTaskRequest(long taskId, bool accept, string path) | accept/reject task execution request |
| void executeTask(string path) | executes downloaded task |
| void writeOutputToFile(string outFilePath) | writes the output of execution to a file located at outFilePath |
| void submitResults(string resultsPath, long taskId) | uploads results of the executed task to the server |

| **Class** | Task |
|---|---|
| Wrapper class for tasks. | |
| **Attributes** | |
| long taskId | global id of the task assigned by the server |
| int userId | id of the user that created task |
| string path | local path to the files of Task |
| **Methods** | |
| Task(int userId, string path) | class constructor |

| **Class** | ServerConnectionManager |
|---|---|

| Object that provides communication between the server and client | |
|---|---|
| **Attributes** | |
| ServerEventHandler server | event handler instance of the server |
| **Methods** | |
| ServerConnectionManager() | establishes connection with server |
| DatabaseToken getPermission(metadata) | asks for permission to submit task |

| **Class** | DatabaseConnectionManager |
|---|---|
| Object that provides communication with the database. | |
| **Attributes** | |
| FileDatabase database | event handler instance of the server |
| **Methods** | |
| DatabaseConnectionManager() | class constructor |
| int uploadFiles(string path, DatabaseToken token) | uploads files related to task to the database. returns the location of the uploaded file in DB. |
| void downloadFiles(string path, DatabaseToken token) | downloads the files on the file database (specified by the token) to the (local) path |

| **Class** | DatabaseToken |
|---|---|
| Token file that gives access to database upload/download actions. | |
| **Attributes** | |
| string address | address in the database where one should up/download |
| string password | corresponding password |

| **Methods** | |
|---|---|
| DatabaseToken(string address, string password) | class constructor |

### 3.2.2 Presentation

| **Class** | LoginView |
|---|---|
| View for logging into the system. | |
| **Attributes** | |
| QLineEdit[5] emailOrUsername | email address or username of the user |
| QLineEdit password | password of the user |
| **Methods** | |
| LoginView() | class constructor |

| **Class** | ForgotPasswordView |
|---|---|
| View for requesting password reset in case of forgotten password. | |
| **Attributes** | |
| QLineEdit email | email address of the user |
| **Methods** | |
| ForgotPasswordView() | class constructor. Sends request for a password reset. |

| **Class** | ResetPasswordView |
|---|---|
| View for resetting the password if it is forgotten. | |
| **Attributes** | |
| QLineEdit newPassword | new password of the user |
| QLineEdit resetPasswordId | uniquely generated password reset ID for authentication |
| **Methods** | |

| | |
|---|---|
| ResetPasswordView() | class constructor. Resets the password. |

| **Class** | RegistrationView |
|---|---|
| View for registration of a new account. | |
| **Attributes** | |
| QLineEdit email | email address of the user |
| QLineEdit name | first name of the user |
| QLineEdit surname | surname of the user |
| QLineEdit username | username of the user |
| QLineEdit password | password of the user |
| **Methods** | |
| RegistrationView() | class constructor. |

| **Class** | ProfileView |
|---|---|
| View for displaying profile information of a user. | |
| **Attributes** | |
| QLabel[6] username | username of the user |
| QLabel name | first name of the user |
| QLabel surname | surname of the user |
| int userId | ID of the user |
| **Methods** | |
| ProfileView(int userId) | class constructor. Displays the profile of the user with the given userId. |

| **Class** | FirstTimeInteractionView |
|---|---|
| View for instructions and quick tips for using the application in first launch. | |

| **Attributes** | |
|---|---|
| | |
| **Methods** | |
| FirstTimeInteractionView() | class constructor |

| **Class** | UsageTypeView |
|---|---|
| View for the selection of usage type; renter or leaser. | |
| **Attributes** | |
| | |
| **Methods** | |
| UsageTypeView() | class constructor |

| **Class** | ListingsView |
|---|---|
| View for listing the available computers to be leased. | |
| **Attributes** | |
| Leaser leasers[] | list of the available leased computers |
| **Methods** | |
| ListingsView() | class constructor |

| **Class** | SpecificationsView |
|---|---|
| View for the specification and configuration of the leased computer hardware. | |
| **Attributes** | |
| | |
| **Methods** | |
| SpecificationsView() | class constructor |

| **Class** | FileUploadView |
|---|---|
| View for uploading files to be executed on the leased computer. | |
| **Attributes** | |
| File files[] | list of files to be executed |
| QLineEdit args | arguments for custom execution and details |
| long taskId | ID of the current task assigned by the server |
| **Methods** | |
| FileUploadView(long taskId) | class constructor |

| **Class** | ProgressView |
|---|---|
| View for the displaying the elapsed time for the rental process and status of the execution. | |
| **Attributes** | |
| long taskId | ID of the current task |
| Time elapsedTime | variable holding the elapsed time duration for the process |
| QLabel status | status of the execution |
| **Methods** | |
| ProgressView(long taskId) | class constructor |

| **Class** | PaymentView |
|---|---|
| View for finishing the session and payment. | |
| **Attributes** | |
| long taskId | ID of the current task |
| int renterId | user ID of the renter |
| int leaserId | user ID of the leaser |

| | |
|---|---|
| double price<br><br>Payment payment | price to be paid for the session<br><br>payment object for finishing the session and handling transaction |
| **Methods** | |
| PaymentView(long taskId, int leaserId) | class constructor |

# 4 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).

[5] Qt Documentation. “QLineEdit Class | Qt Widgets”. https://doc.qt.io/qt-5/qlineedit.html

[6] Qt Documentation. “QLabel Class | Qt Widgets”. https://doc.qt.io/qt-5/qlabel.html