Faculdade de Engenharia da Universidade do Porto

Enterprise Distributed System

Distributed and Integration Technologies (TDIN) 2019/2020

Authors:

Carlos Daniel Coelho Ferreira Gomes	up201603404
João Carlos Cardoso Maduro	up201605219

Architecture **5**
Functionality 5
Modules 6
Interactions 7
List of Functionalities **8**
Worker Register 8
Worker Login 8
Worker Logout 8
Issue Creation 8
Issue List 8
Email Reception 9
Solver Login 9
Solver Register 9
Issue and Question Listing 9
Assignment of Issues 9
Rising Questions to an Issue 9
Solve Issues 10
Department Solver Login 10
Department Solver Register 10
Question Listing 10
Functionality Tests **11**
Worker Register 11
Worker Login 11
Worker Logout 11
Solver Register 11
Solver Login 11
Solver Logout 12
Issue Creation 12
Email Reception 12
Unassigned Issues Listing 12
Assigned Issues Listing 12
Assigned Issue Update 12
Make a Question While Unassigned 12
Send Question to the queue 13
Message persistence in the queue 13
Main Flows of Work **14**
Build Instructions 24
Configuration of the Services and the Applications 24
How to use the System 24
Bibliography and References **25**

Architecture

Functionality

With our implementation of a distributed system for a company to use to solve their issues, it is possible for workers of the same company to publish their issues to be solved by solvers.

These workers can track the evolution of the state of the issue through a dashboard, and which information is updated in real-time. The possible states for their issues are “unassigned”, “assigned”, “waiting” and “solved”. Also, when an issue is solved, the workers will receive an email notifying them about the solution to their issue.

When a new issue is created, they are sent to the server, where are stored in a database, initially in the “unassigned” state

When the solvers start their GUI applications, the applications will fetch all the unassigned issues in the server database, and display them in the “Unassigned Issues” list. If new issues come after that, the server will notify the solver applications, and add them to the list. The solver can now proceed, to click in an issue, which will open a window with more detailed information about the issue, a button to the solver be able to self assign to the issue, a button to emit secondary and a text box to write the answer to the issue. These last two features only work after the solver was assigned to the same issue. When the solvers assign to an issue, the card representing it will shift to the “My Issues” List, and the “Assign” button changes to a “Solve” button. After assigned, the application notifies the server about that, and the state of the issue is updated to “assigned”, while the assignee field of the issue is filled with the solver email. The other solvers are notified about the assignment of the issue and remove it from their lists. If the solver writes an answer and presses the “Solve” button, the same answer is sent to the server, where its state is updated to “solved” and the worker, who created will be properly notified, while the card from the list is removed. But if the solvers choose to make questions to a department, they can press the “+” button to write a question and choose to which department send it.

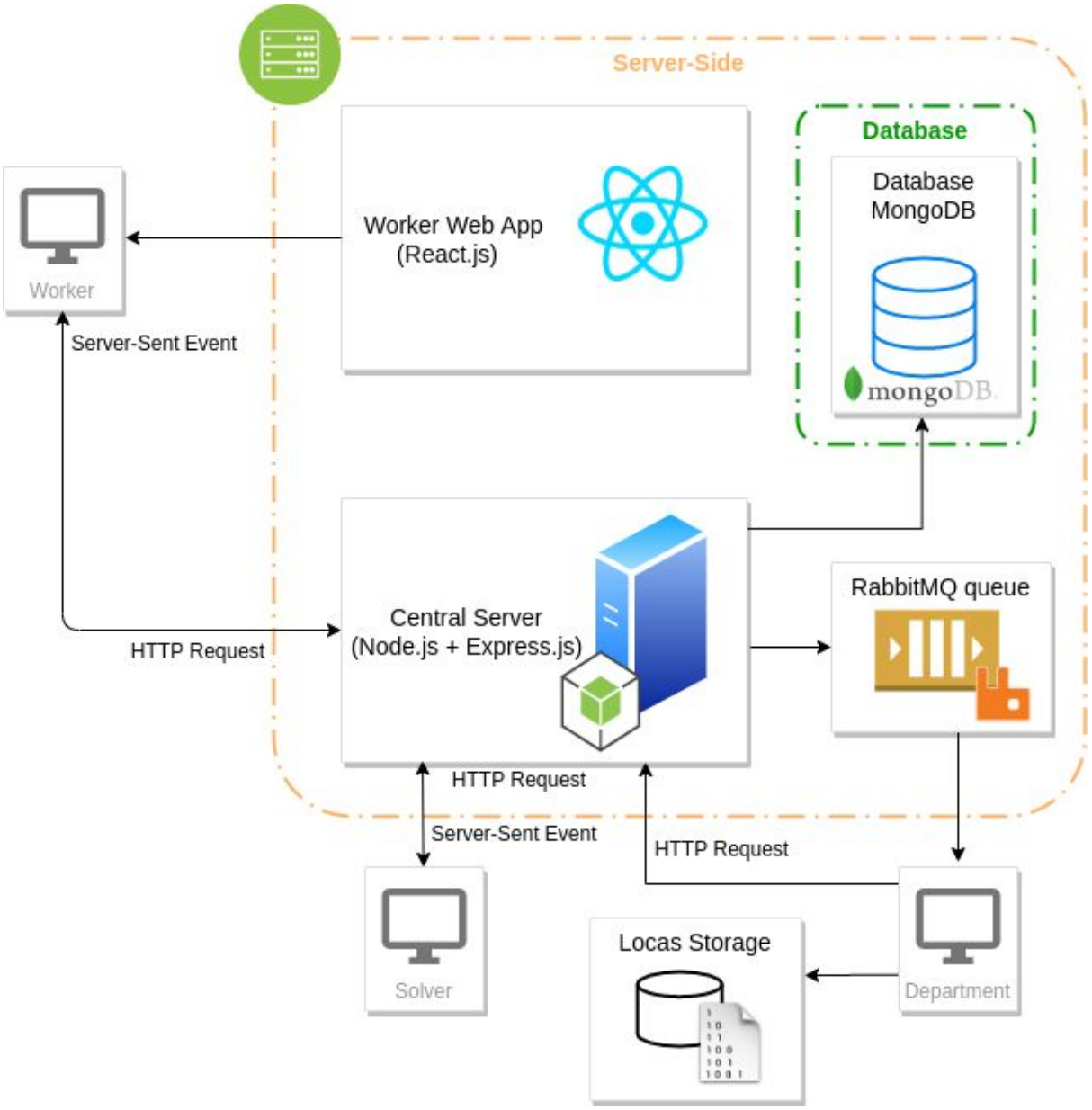
After publishing the question, it will be sent to the central server, which will store it in its database and will forward it to the department queue. The central server also updates the state of the issue to “waiting” and links the issue with the recently created unresolved question.

All departments have their own queue, and every question sent to the queue will persist until one solver of the department associated with the queue, opens their GUI application. The application loads the issues in the queue, and if they are closed, those issues are persisted locally and can be loaded again after reopening the application. After the solver of the department, presses one of the questions, a window to answer the question will open. After answering the question, the application will send the answer to the server and the question is erased from the list.

All the users (workers and solvers), must authenticate to start using the services. Once authenticated, they are granted a JSON Web Token, which contains encrypted essential data about the users, like their emails and their roles. Every request sent by them

is evaluated by a middleware present in the central-server, in order to extract the email and the role associated with the token of the request. If the token is not valid, the user is not valid or the user does not have the necessary role for the requested operation, the request is rejected.

Modules



Our solution to this project is based in a microservices architecture, and the existing services are the worker web application server worker-app, the central-server, the message broker RabbitMQ where the queues are hosted, and the service mongo which host the MongoDB database.

It is possible to run all those services using docker containers, to make their integration easier for everyone to deploy, without having to install the dependencies.

To interact with these services, there are the worker-apps, which are used by the workers and are web applications rendered by the web application server; the solver-apps which are used by the solvers and the department-apps that are used by the department solvers.

The worker-app is a Node.js server that renders the web application the workers, using the React Framework.

The central-server is a server running using the Node.js runtime with the Express Framework.

Aa NoSQL MongoDB database is hosted in the service mongo. The database has 3 schemas: User, Issue, and Question.

The message broker RabbitMQ is responsible to host the department queues.

Both the solver application and the department application use the .NET Core runtime with the GTK# library for the GUI.

Interactions

All communications between all the entities described are made using the HTTP protocol. The services to communicate with each other using the REST architecture. To contact the services, the client applications (both web and desktop GUI) also use REST.

To notify the client applications that subscribed to the central server, the Server-Sent Events (SSE) technology is applied, except the case of the Department Applications, where the questions are published first into the queue and then received by the application listening to it.

The worker web applications are rendered by the Worker Application Server. These applications then contact the central server using all the endpoints that start with “/api/worker”. These endpoints are used to create issues.

The solver applications contact the central server using the endpoints that start with “/api/solver”, in order to fetch the actual unassigned issues, update their statuses or emit questions.

the Central Server listens to the received from the worker web applications, the solver desktop app, and the department desktop apps. It also reads and writes from the Mongo Database. When a Question is created, this server publishes it into the queue of the department specified in the question. To notify both worker apps and solver apps, the server sends Server-Side Events (SSE), but first, the applications need to subscribe to them using the endpoint “/api/stream”, in order to establish a channel with the server to receive the events. These connections are mapped to the user email. All the request sent to the server, except the ones related to authentication, need a JSON Web Token, in order for the server to know the identity of the author of the request and to check if they have the role associated with the requested action.

The department app
only listens to its department's queue. When a question is solved it is sent to the central server.

List of Functionalities

Worker Register

If someone wants to makes a question to de Enterprise System he can register, inserting username, email, password, and confirming this password. If the email is not in the database the user will be registered and sent to the login page. In the server-side, this user as the role “worker” that is used for authentication and authorization.

Worker Login

If a user is already registered in the system, he can insert its credentials, email, and password, and if they are correct, he will be able to login in the system. Otherwise, he will be kept in the login page. When the login is successful, the worker receives a token that is used for authentication and authorization on the server-side. It is sent in every request to the server, without it the worker is logged out. If the user loose this token, he is logged out and redirected to the login page.

Worker Logout

A worker that is in the issue page can logout pressing the button logout. He will be redirected to the login page.

Issue Creation

Only a worker can create an issue. In order to do it, he needs to be logged in. Then, pressing the button “New Issue”, a modal pops up and he can insert the title and the issue description. The issue can be sent only when the user fills the description and title fields. If the issue is created on the server-side, it is inserted automatically in the Issue List.

Issue List

When the worker is logged in, he can only access to a page where there are all his issues. Each issue is listed with: title, state (unassigned, assigned, waiting for answers or solved), description, date, and time of creation, and when the issue is “solved”, the solution is presented too. The resolution state of the issue is automatically updated when some change to its state happens.

Email Reception

When a solver solves an issue the server sends an email to the worker, using the email "tdin2020solver@gmail.com" to send the email to the worker email (given when registering). We use the Gmail service to send the email.

Solver Login

Just like a worker, the solver receives a token when login is successful. To a login be successful, the solver has to be register in the system, with the correct credentials that he insert to do this action.

Solver Register

Just like a worker, a solver can be registered with an email, full name, and password.

Issue and Question Listing

In the main window of the solver, there are two lists, assigned issues, and unassigned issues. He can double click on an issue and a window pops up. In an unassigned issue, he can open it and see the issue title and description and assigned himself to it. In an already assigned issue he can solve it, writing the resolution or raise some questions, and sent them to the selected department.

Assignment of Issues

Only a solver can be assigned to an issue, solve an issue, or raise questions to an issue. When a solver logs in, he receives all unassigned issues in the system and all issues he is assigned to. When a solver assigns himself to an issue, all the active solvers loose immediately this issue from their own list of unassigned issues.

Rising Questions to an Issue

When a solver is assigned to an issue, to help him he can raise questions to the department "issues" (in the project assignment the teacher says to consider only one department). This issue goes to a queue that the department solver is listening to. If the department solver is not up, the question remains in the queue until a department session. When the department solver logs in, he reads the queue and can answer the question. When a question is solved, the answer is immediately sent to the solver that raised it if this is up.

Solve Issues

Only a solver can solve an issue and he needs to be assigned to it. The solving of an issue can only be done when there is no question unanswered. When an issue is solved, it is removed from the list of issues of the solver.

Department Solver Login

A department solver has a solver role. The authentication and the authorization processes follow the same rules of Solver and Worker.

Department Solver Register

This registration follows the same rules as the other users.

Question Listing

When a department solver does login, he reads the queue all the questions. These questions are listed and the department solver can double click and answer the questions. When it solves the question, it is directly sent to the server and the solver that raised the question is automatically notified.

Functionality Tests

Worker Register

In order to test register, different combinations of nicknames and passwords were introduced. We verified that an already existent username makes the registration action fail. A non-existent username but with a password that not follows the rules makes the registration fail too. Only a non-existent username and a password (equal in the confirmation field) following the rules makes the registration be successful.

Worker Login

In order to test Login, different combinations of nicknames and passwords were introduced. If the password does not correspond to the nickname the login fails. If the nickname does not exist, the login fails. The login only retrieves success when the nickname is an existent one and the password is correct.

Worker Logout

To test logout, we tried to logout a worker. After that, the user need to login again to see his issues again.

Solver Register

In order to test register, different combinations of nicknames and passwords were introduced. We verified that an already existent username makes the registration action fail. A non-existent username but with a password that not follows the rules makes the registration fail too. Only a non-existent username and a password (equal in the confirmation field) following the rules makes the registration be successful.

Solver Login

In order to test Login, different combinations of nicknames and passwords were introduced. If the password does not correspond to the nickname the login fails. If the nickname does not exist, the login fails. The login only retrieves success when the nickname is an existent one and the password is correct.

Solver Logout

To test logout, we tried to logout a solver. After that, the user need to login again to see the lists of unassigned issues and assigned issues.

Issue Creation

We tried to create an issue like a normal worker and after submitting it, we confirmed that the solvers' lists were updated in their GUI, confirming that both the creation of issues and subsequent notifications were working.

Email Reception

After solving an issue, we checked that our mailboxes did indeed received an email with the answer, confirming that the email reception was working.

Unassigned Issues Listing

After login in a solver application, we confirmed that all the unassigned issues listed matched the ones in the database that were unassigned too.

Assigned Issues Listing

After login in a solver application, we confirmed that all the assigned issues listed matched the ones in the database that were unassigned to the logged user.

Assigned Issue Update

After assigning to an issue, the same issue shifted from the "Unassigned Issues" list to the "My Issues" list. All the other applications running at the same time, had the issue removed from their lists.

Make a Question While Unassigned

We were not able to add a question to an issue unassigned.

Send Question to the queue

After making the question, using the RabbitMQ dashboard we were able to track the arrival to queue and the reception to the department solver.

Message persistence in the queue

Using the RabbitMQ dashboard, we were able to check that a message does not get lost if the department application is not running.

Main Flows of Work

Firstly the user should follow the instructions to build and run all the services. Then for a worker you should open your browser and go to “localhost:8080” and login with an account or register yourself.

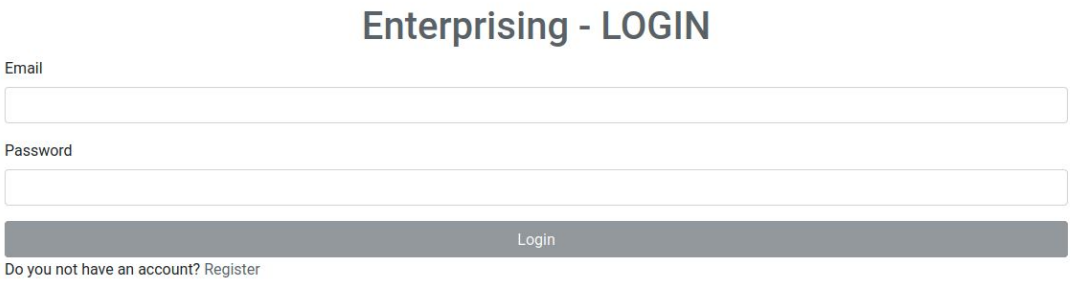


Fig.1 - Login Page

Fig.2. Register Page

After the login success (if you register first you need to login to enter) you will be presented in your Page.

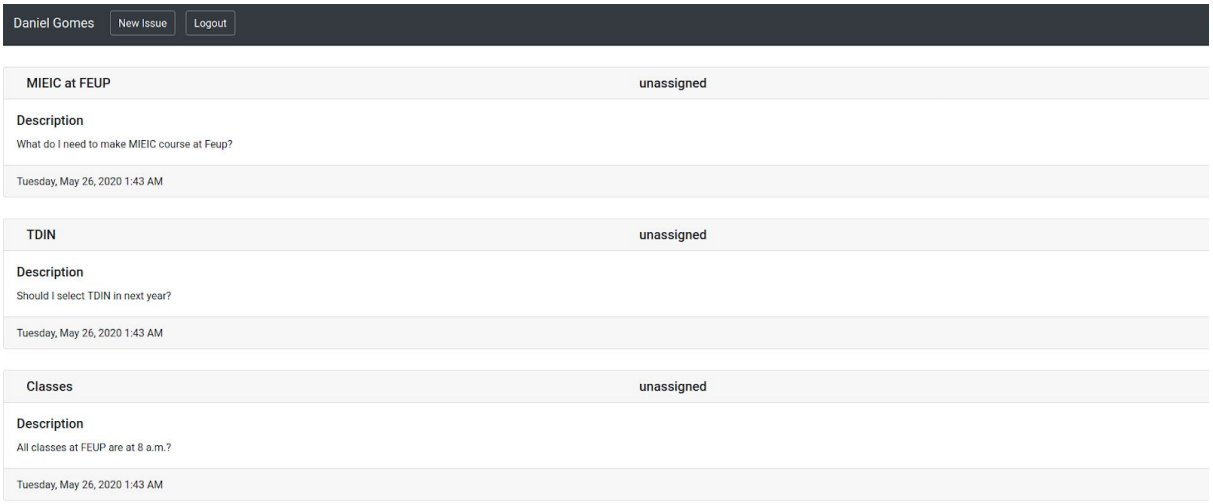


Fig.3 - Main page (this user has 3 issues already)

Here you can see your own issues and their state. You can create your issue pressing “New Issue”:

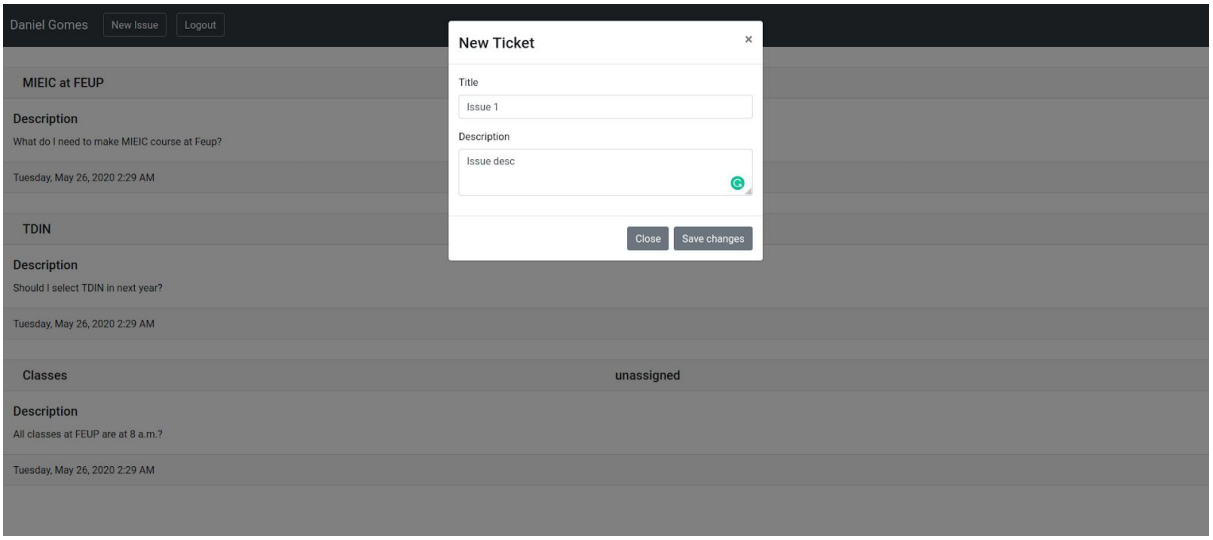


Fig. 4 - New Issue Form

After the submission, the issue will be added to the list. Now you just have to wait for the solvers reply

In the Solver, you have to login as a solver or register (see User Manual for login credentials) and you can assign, answer issues and rise questions.

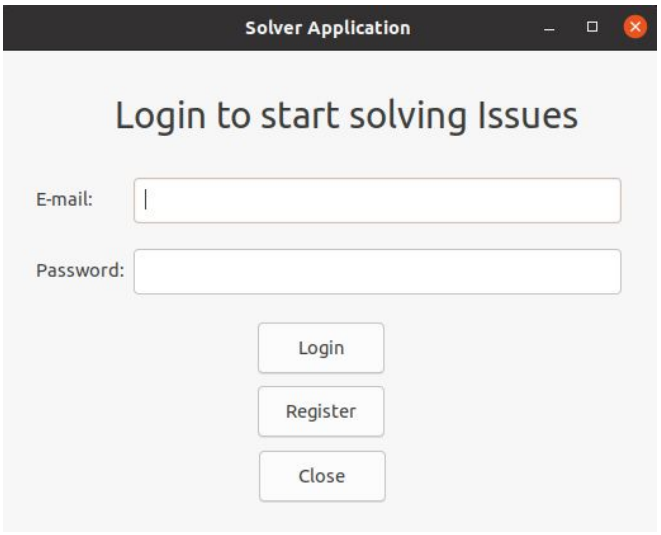


Fig. 5 - Login as Solver

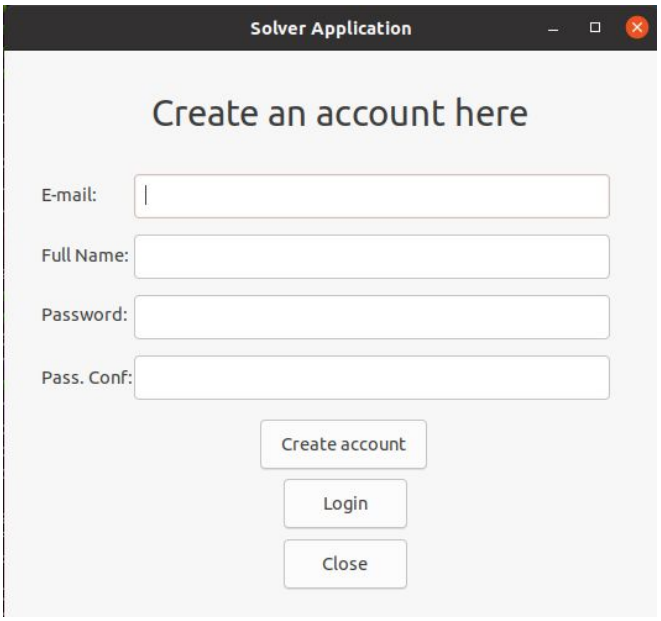


Fig. 6 - Register as Solver

After the login, you can assign yourself to issues and answer them. There are two lists with your already assigned issues and issues that have no assignee.

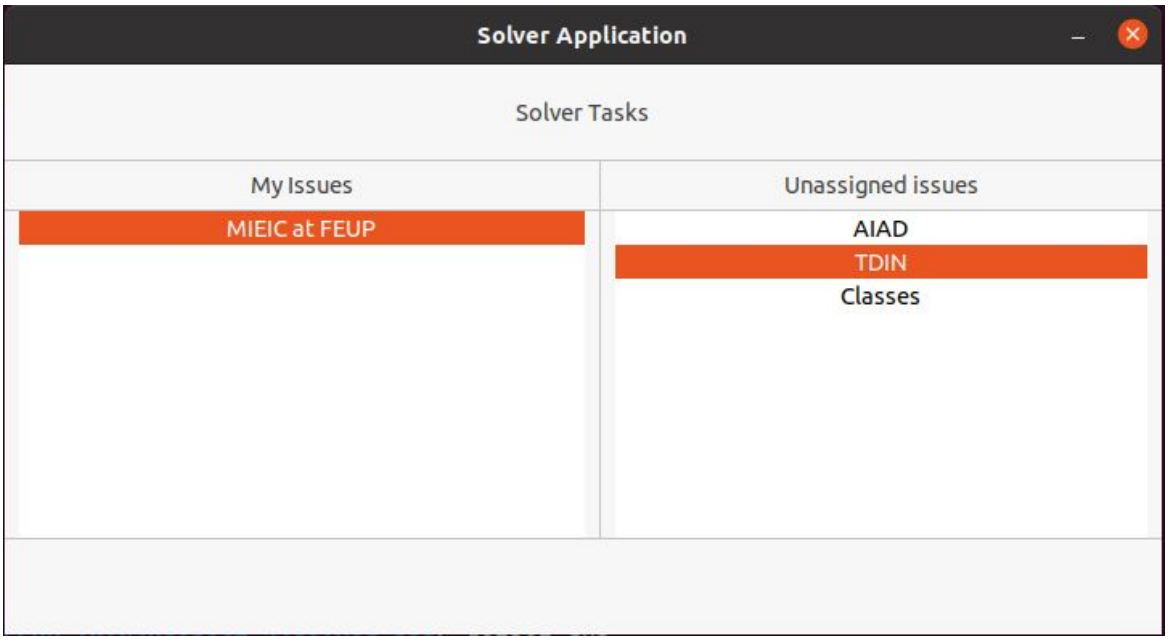


Fig. 7 - Solver main window (with 1 issue already assigned)

Now you can press an unassigned issue and it pops it window with all informations to you decide if you want to assign yourself to solve it.

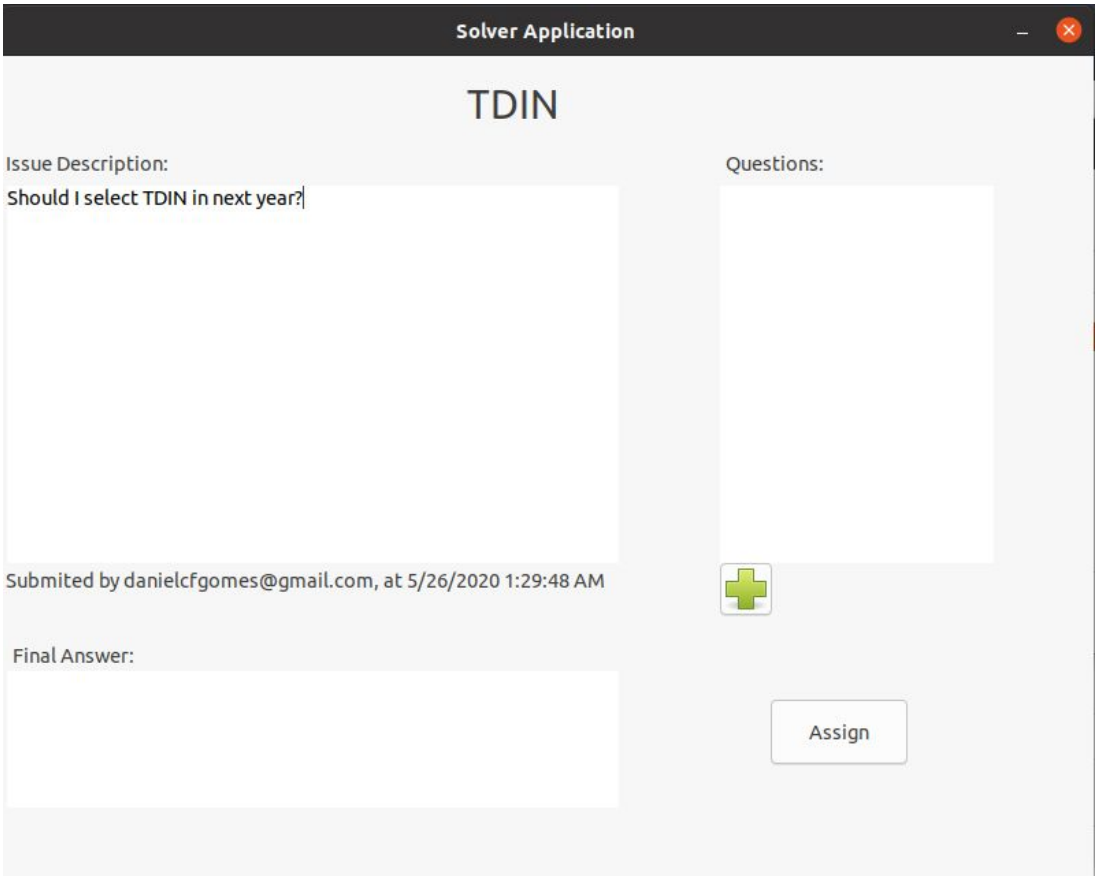


Fig. 8 - Issue window

Pressing the assign button you are able to answer the issue or rise questions. When you press to assign the issue, the worker who created the issue is notified (figure 10).

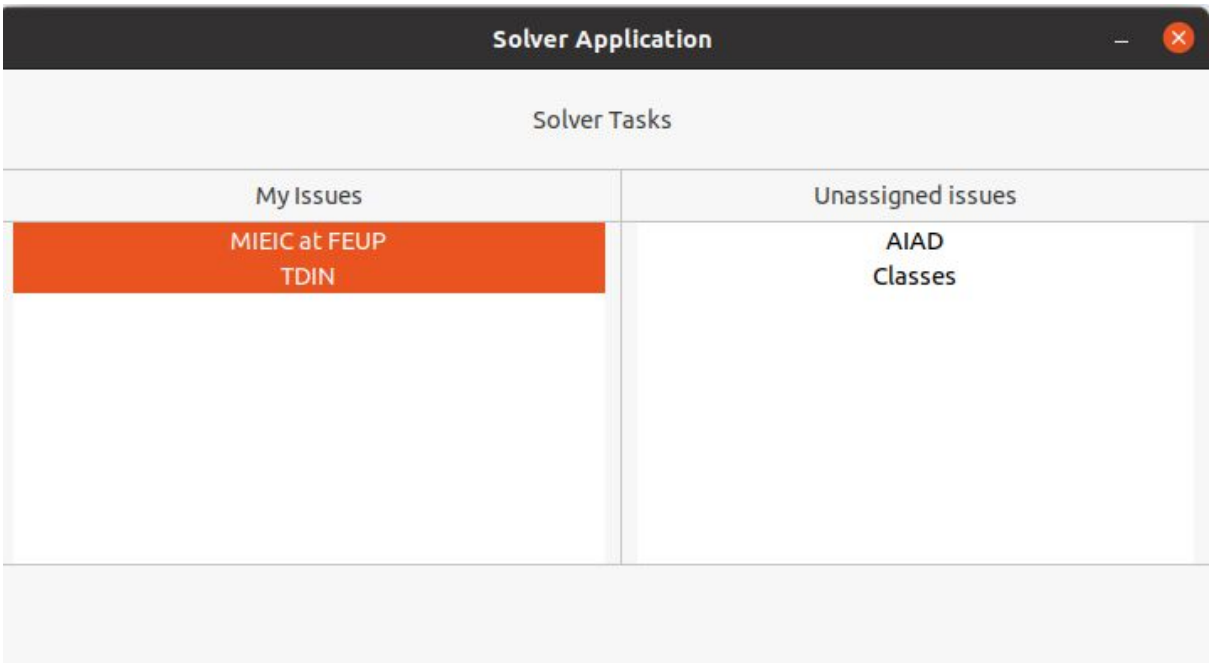


Fig. 9 - Issue assigned to myself

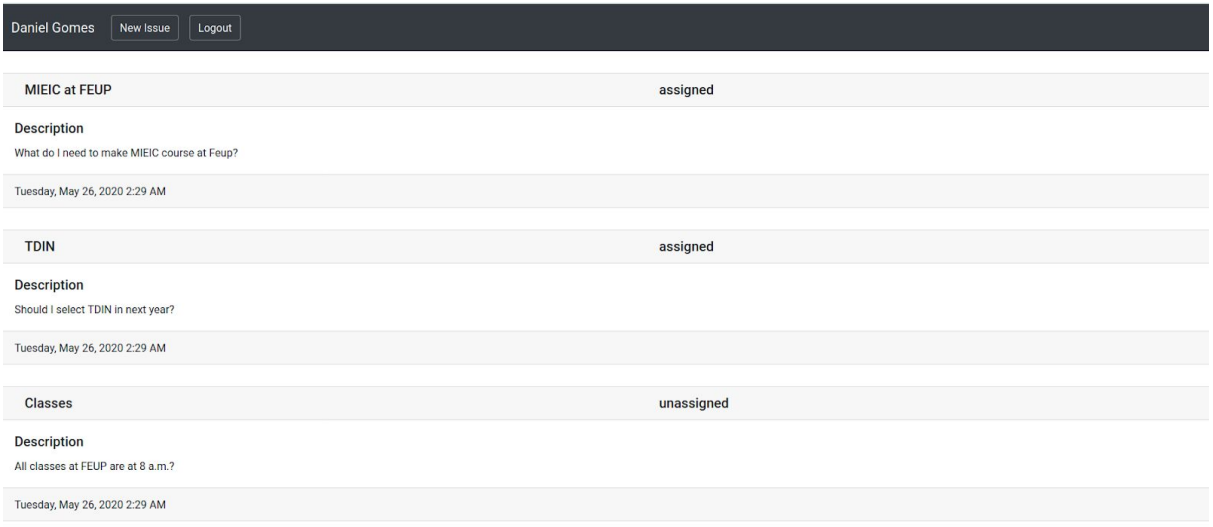


Fig. 10 - Worker notified about the assignment

In order to solve the issues, the solver can also raise questions (figure 11) and need to wait (“asynchronously”) for the answers to solve the issue.

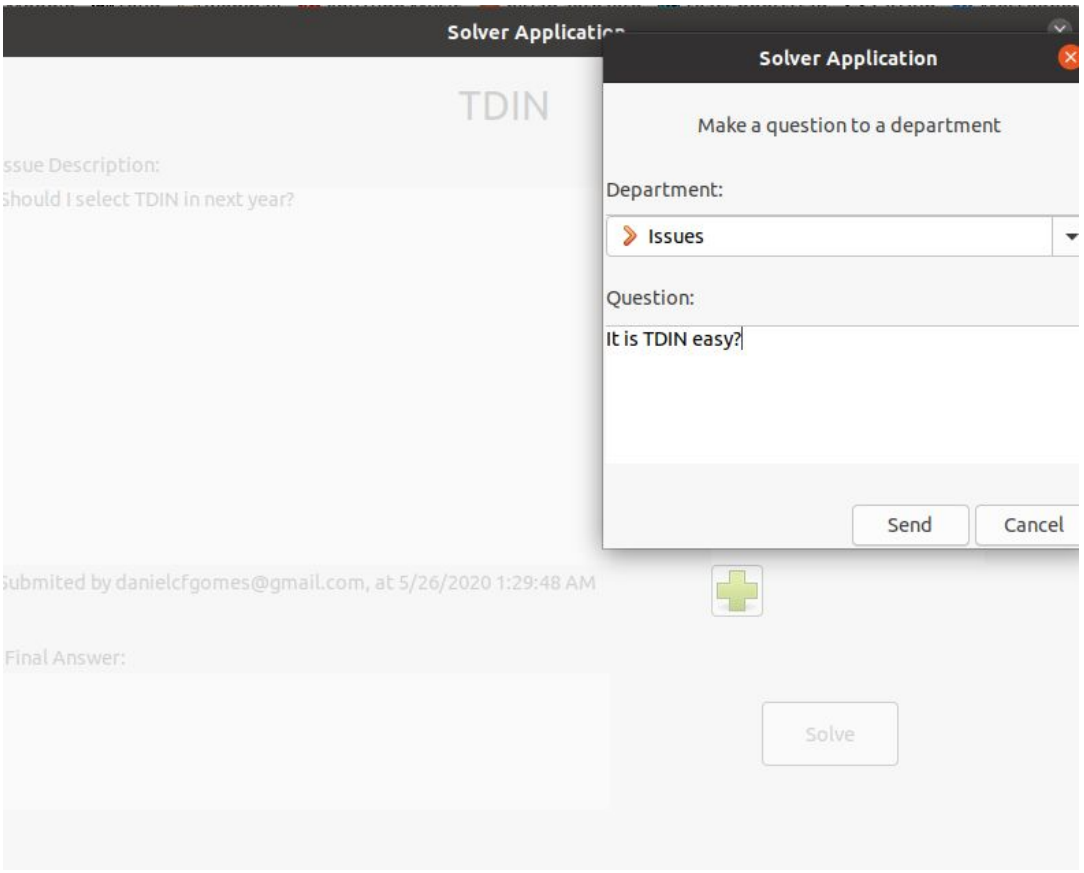


Fig. 11 - Question Form (shows after press “+”)

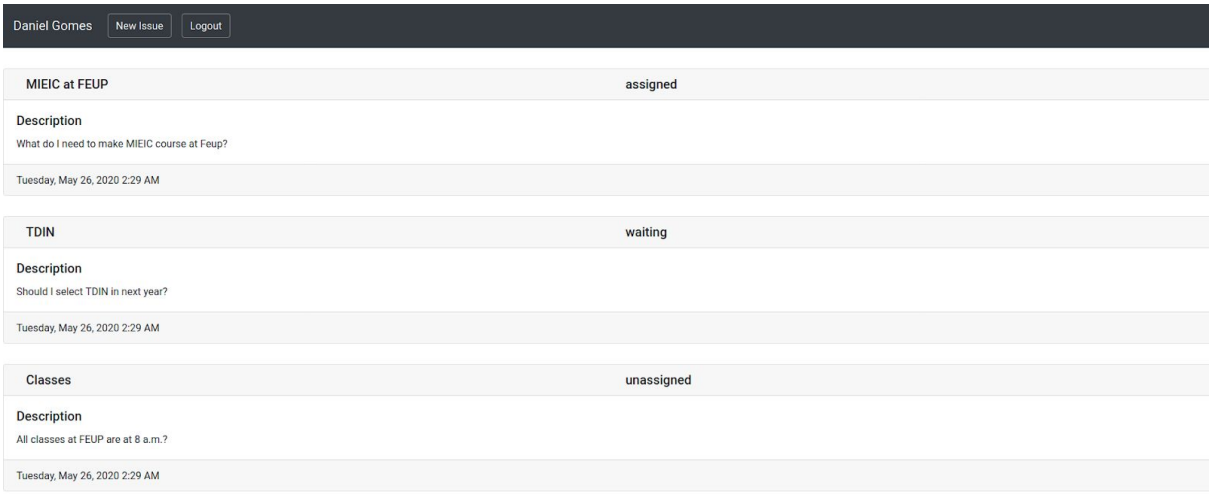


Fig. 12 - User notified with waiting (the solver is waiting for an answer)

Now, log in the department app (same UI design) and check there are questions (figure 13). You can press one question and answer it (figure 14).

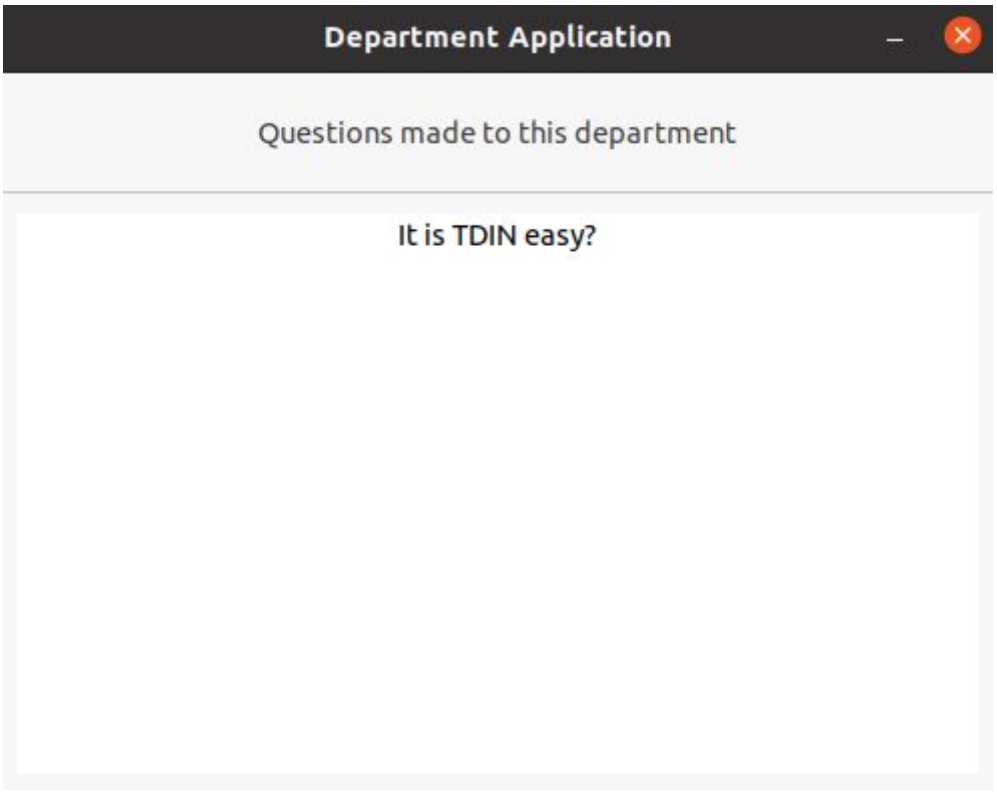


Fig. 13 - Department app view with unanswered questions

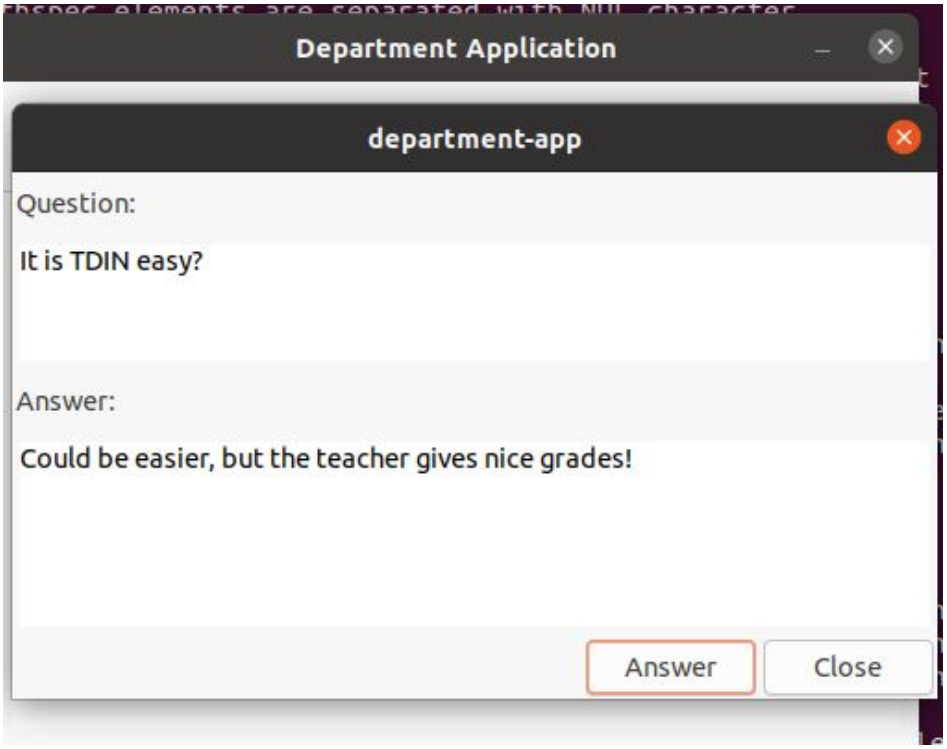


Fig. 14 - Answer to a question

So now, move on to the solver and you can check that the answer will be there (figure 15) and now you are able to solve the issue (figure 17; as you solve the issue it won't be in the window) and notify (what is done automatically - figure 18) the worker.

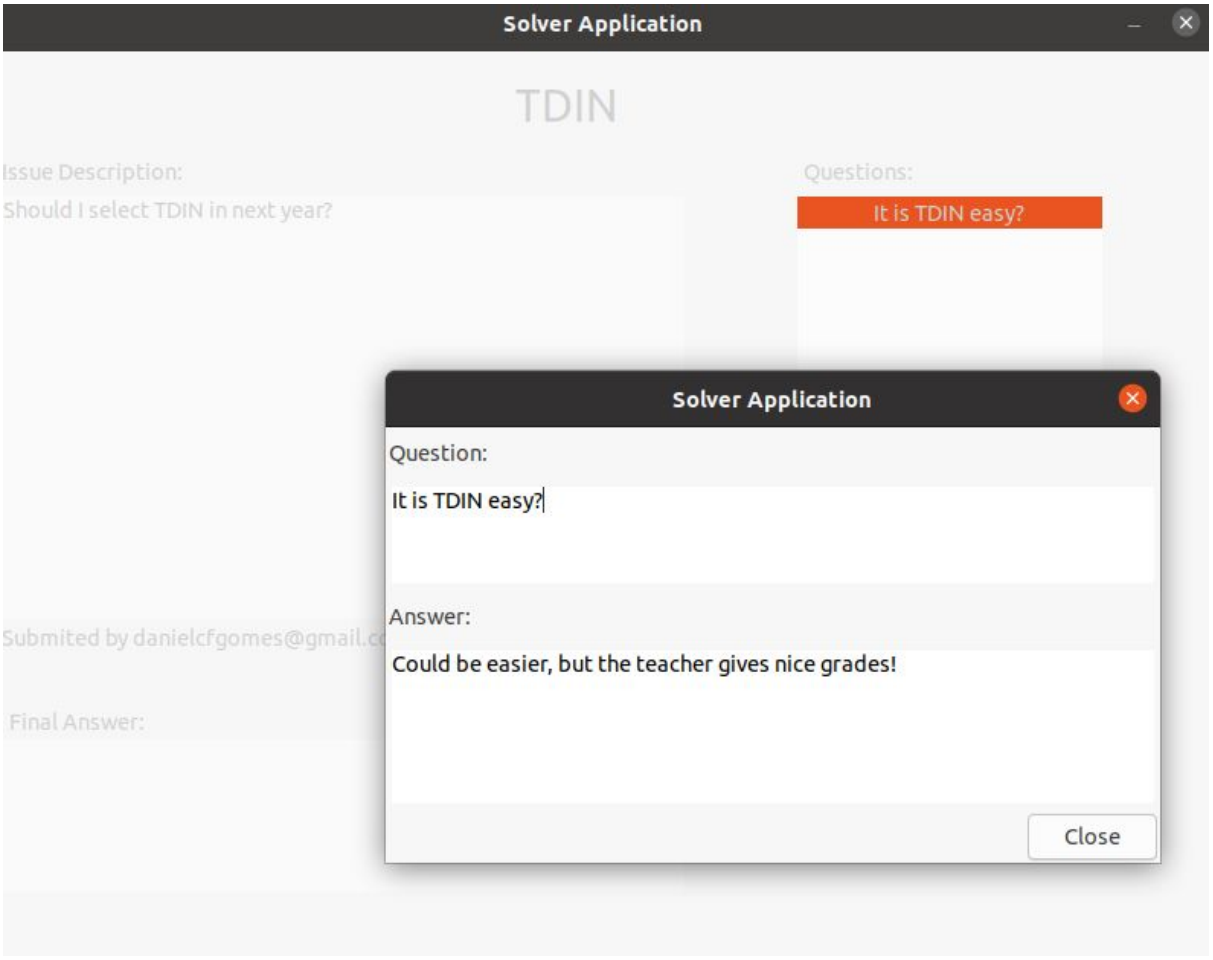


Fig. 15 - Solver got answer automatically and is able to solve the issue

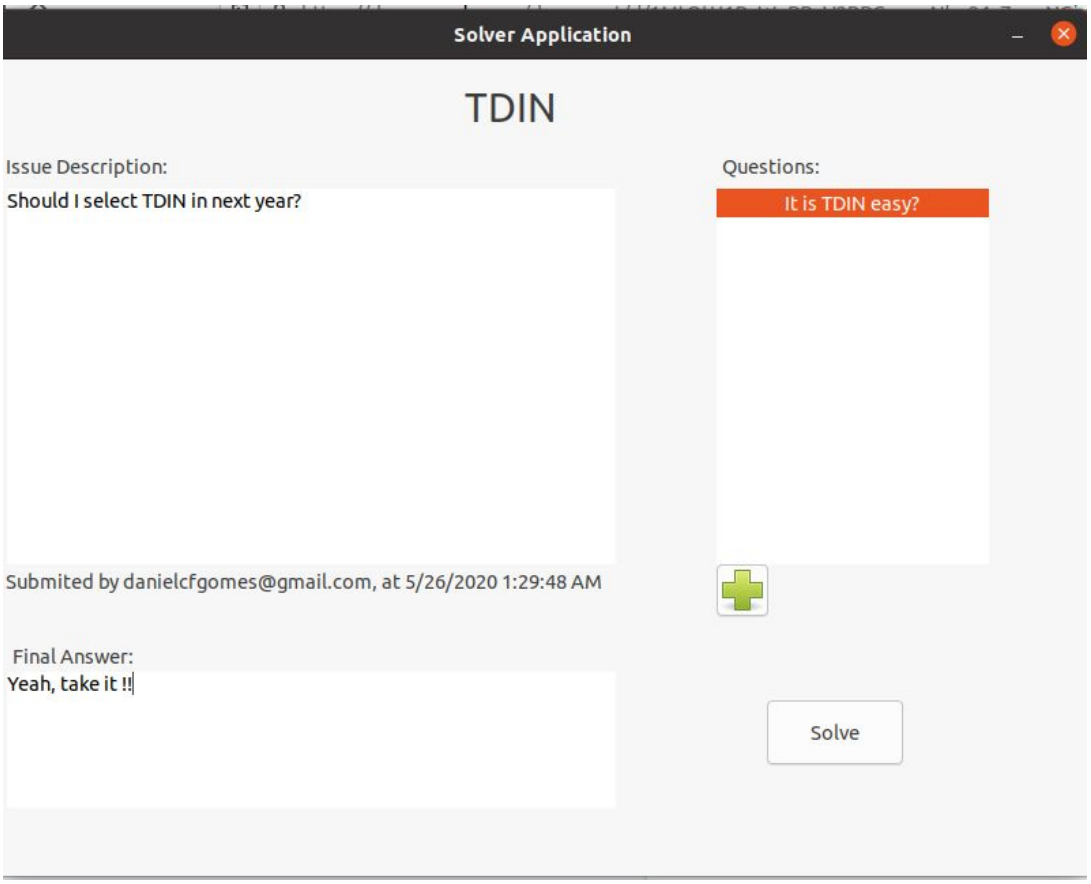


Fig. 16 - Issue ready to be solved

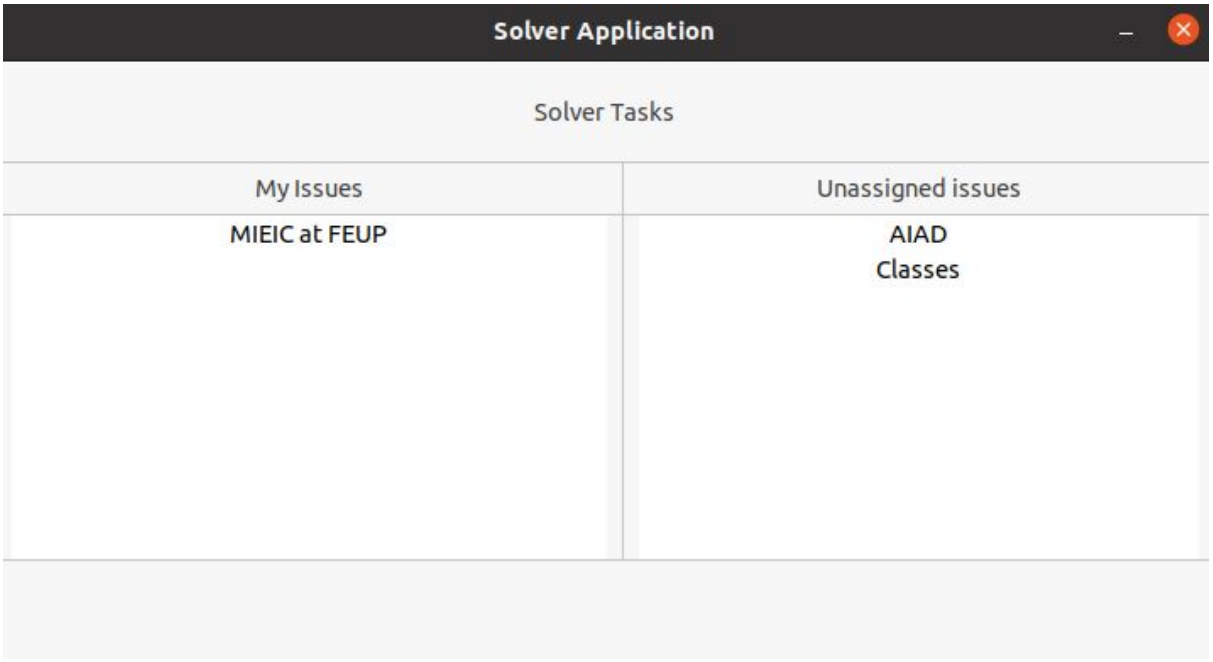


Fig. 17 - Issue Solved

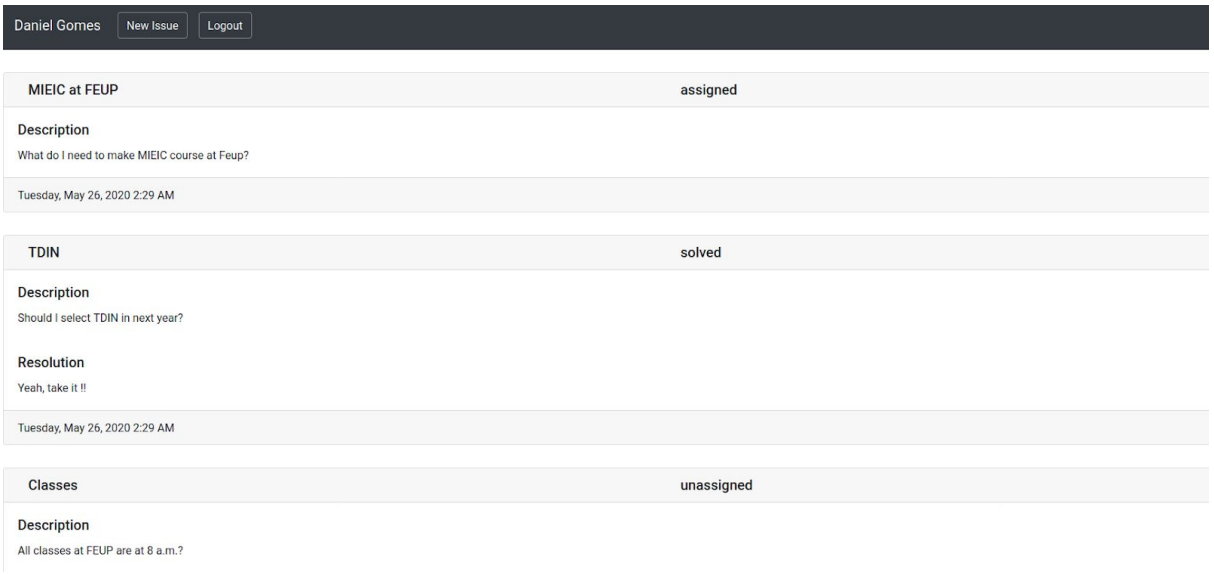


Fig. 18 - User notified

You can also check the email of the worker if you received the confirmation email (figure 19).

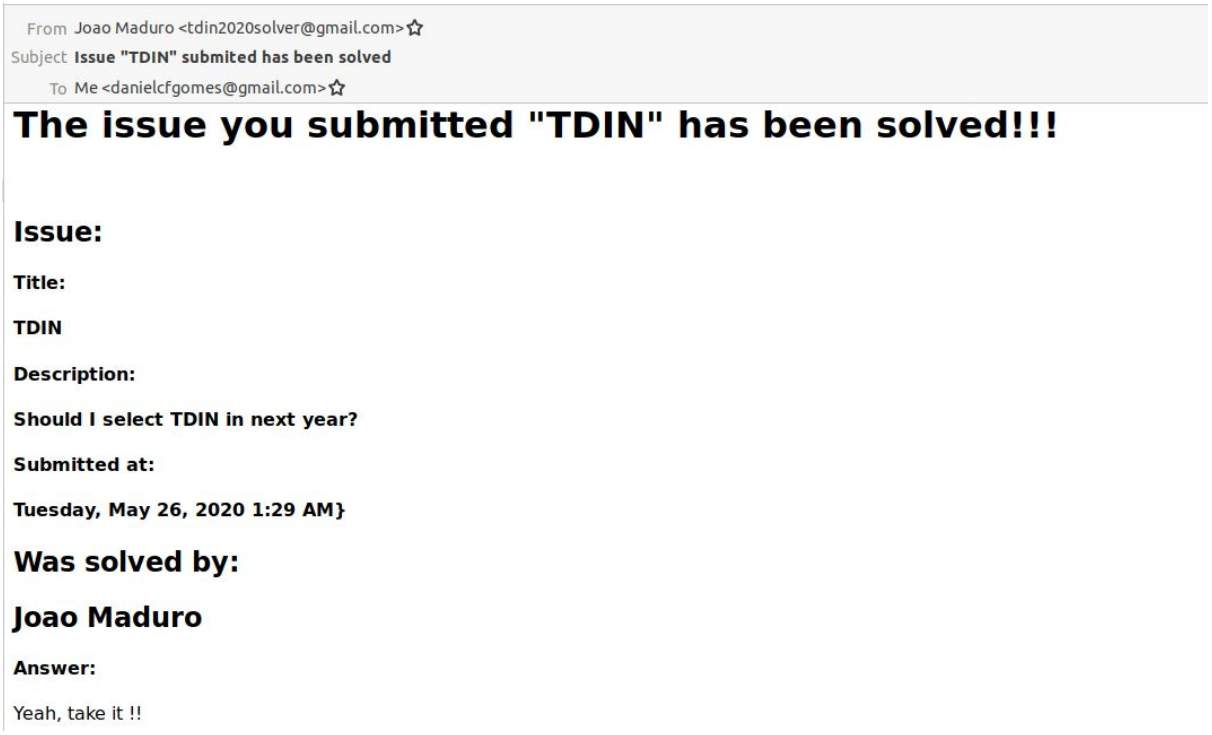


Fig. 19 - Email received

User Manual

Build Instructions

To build all the services first you need to have Docker installed in your machine to be able to build the containers. Go to the root of the project, where the docker-compose.yml file exists and run the following command in the command line: “docker-compose build”. After that all the containers and respective dependencies are built and ready to be used.

To build the solver-app and the department app need the .NET Core runtime and the GTK library installed to build them. To build the executables, run in the folder of each application (/solve-app and /department-app), the following command: dotnet build --configuration Release. After that the release executable is present in the bin/Release/dotnetcoreapp folder. You can also use an IDE like Visual Studio to build the release executable. Then copy the “.env” files to the folder of the release

Configuration of the Services and the Applications

All the variables needed to configure the system are located in environment files (.env), present in the root folder of each entity. The default configuration that comes with them is enough to run the system, but if you want you can edit some variables like the address of the services for example. If you build the release, don’t forget to copy the “.env” files to the folder of the release.

How to use the System

To run the services just run the command “docker-compose up”, in the root of the project, where the docker-compose.yml is located.

The web application of the workers can be found in “http://localhost:8080”.

To run the GUI applications (solver and department), you can run the executables found in the bin/Release/dotnetcoreapp folder of each application. Make sure to have all the services running, otherwise, you are unable to use the applications.

The database is automatically populated with issues, with a worker (danielcfgomes@gmail.com) and a solver (joao@mail.com). Both have the same password which is “123456”

Bibliography and References

- TDIN classes slides: https://paginas.fe.up.pt/~apm/TDIN/
- rabbitmq documentation: https://www.rabbitmq.com/getstarted.html
- Express.js documentation: https://expressjs.com/en/4x/api.html
- Mongoose documentation https://mongoosejs.com/docs/api.html