Documentation

“Progetto di Ingegneria Informatica 2021/2022”

Tutor: Professor Giovanni Agosta

GitHub: *https://github.com/zibasPk/PoliTutorBot*
Video Demo: https://youtu.be/Xnlc61NzWW4

Milo Brontesi
Ingegneria Informatica, School 3I, Polimi
C.P. 10704659

Index:

1. Introduction

The project consists of a telegram bot to facilitate the work of the tutoring administration office, it allows an easy and intuitive way for students in need, to look for, and reserve a match with a peer-to-peer tutor.

The Idea of doing a bot came from a series of exchanges between the tutoring secretariat and the PoliNetwork student association. It was decided that the task could be assigned to a student who was supposed to do the “Telegram Bot” project with Professor Giovanni Agosta.

1.1 Requirements and constraints

The bot is supposed to permit a user to choose the most appropriate tutor for his needs.
To show the most suited tutors, the bot needs to acquire information about the student who makes the request.
For privacy reasons the bot can’t access the student’s data, it has to ask for it explicitly.

Beyond the career data (school, course, exam) needed for showing the appropriate tutors, the user needs to tell the bot their student number. This is needed to save the reservation entry and to check if the student has the right to have a tutoring (not all students have the possibility to get a tutorage).

A tutor is shown to the user only if he has an available slot, each tutor generally has 1 slot per exam but this can be changed at the discretion of the secretariat.

To hinder erroneous reservations and spam attacks a telegram user needs to be locked for a set amount of time after reserving a tutor. A user is also locked if the associated person code is already in an active tutoring.

1.2 Work in progress and future implementations

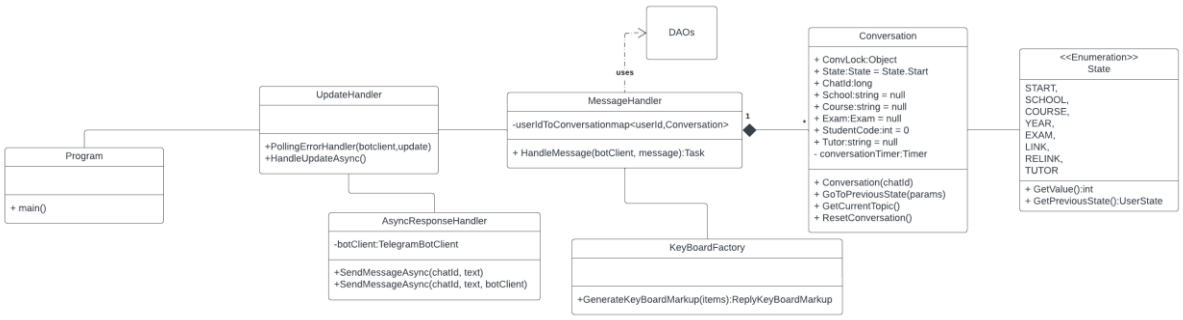
The bot needs a Web site to manage the bot database. In a recent call with the secretary it emerged that the administration may want to use the site for all administrative purposes for the peer-to-peer tutoring.

Currently the bot has an attached web API with various authenticated endpoints which are needed for DB back end management.

The future development for the bot will continue as an extracurricular activity through the “150 ore” initiative.

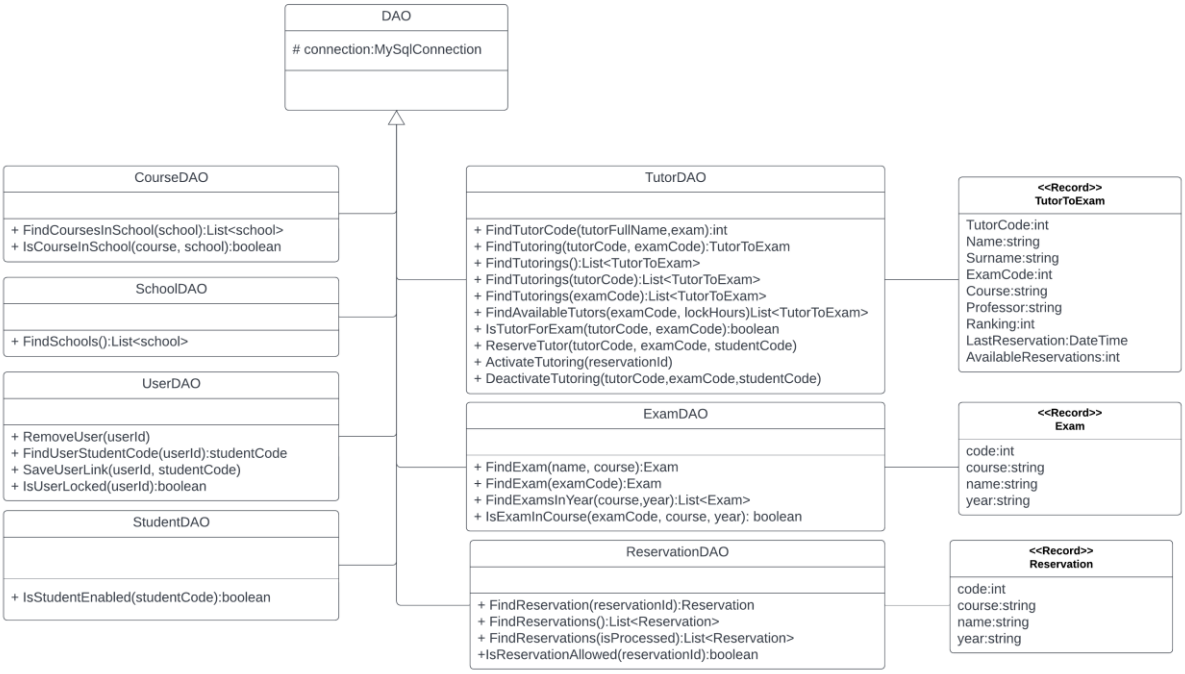
2. Architecture

2.1 Class Diagrams

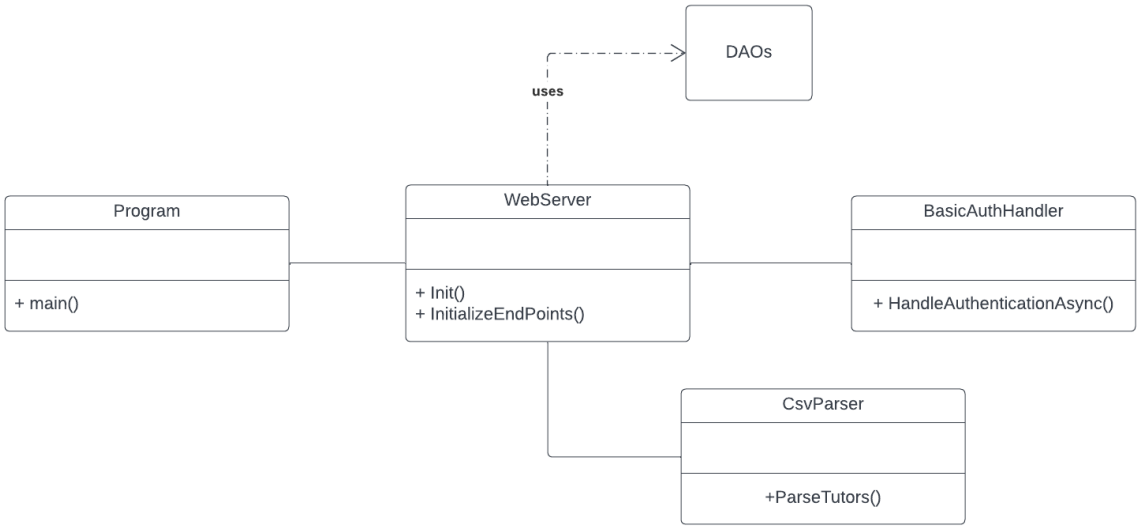


The bot works by utilizing the telegram bot API library, the connection with the API is established in the **Program** Main() method where a update handler is assigned.

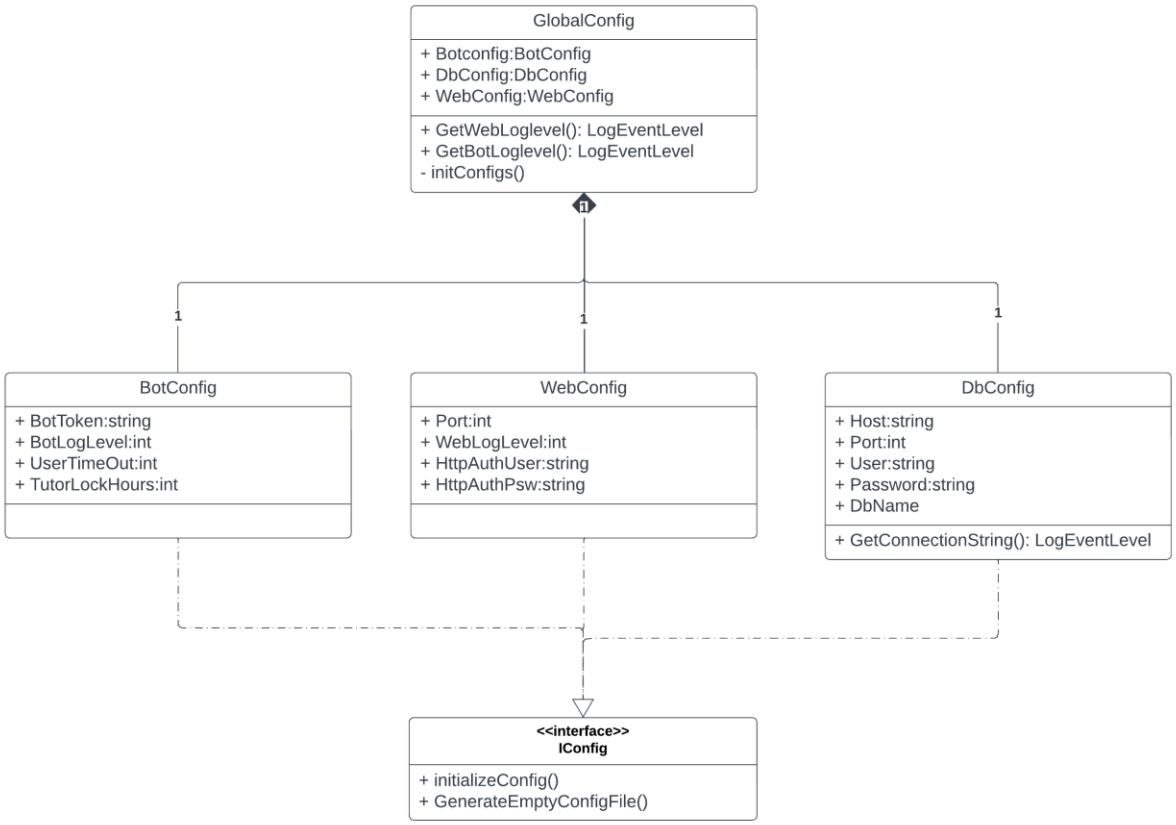
The **UpdateHandler** class can manage various update types (message, edited message, callback …) via specific handlers, but in our case only messages are processed. These messages are handled by checking the State of the conversation associated with the user ID of the sender. Depending on the State only certain messages are accepted and in relation to the received message a response can be sent back.
Responses are a direct consequence to a message, to allow sending messages to a user asynchronously there is the **AyncResponseHandler** class.



A message is checked and its function is processed by making queries to the DB. These queries are made by Data Access Objects (DAO) and information is retained in the appropriate data classes (records).



In the Main() method the WebServer is initialized. All http endpoints require authentication to answer which is handled by the **BasicAuthHandler** class.

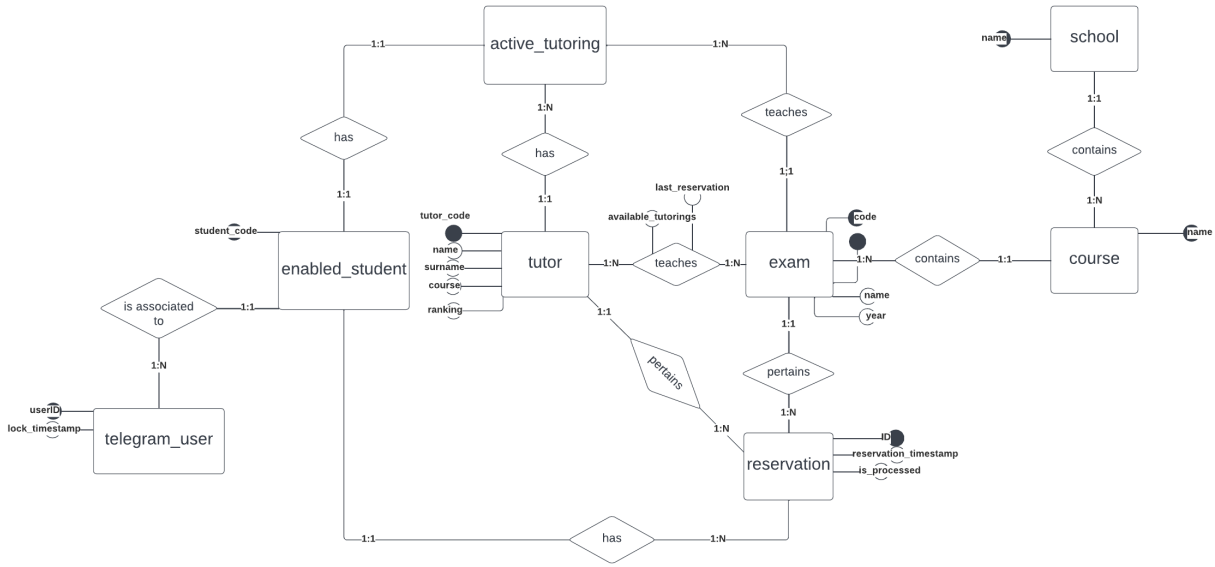


The **GlobalConfig** class is used throughout the application and contains all customizable variables of the bot.

All configuration classes are initialized on startup. A configuration tries to load information from a JSON file, if the file isn't found it creates it.

2.2 Data Base Design

The database is designed to accommodate present and future requirements, it was implemented in MySql Workbench.



2.2.1 *ER diagram*

school, course, exam:
entities that are necessary to show the available career data that a user can insert.

tutor:
entity that keeps track of the data regarding a tutor.

active_tutoring:
entity that contains an active tutoring (a tutoring is activated by the secretariat).

reservation:
a tutoring reservation, the **is_processed** attribute is used to identify reservations that have been elaborated by the secretariat.

enabled_student:
a student who is eligible to tutoring.

telegram_user:
a saved telegram user, the **lock_timestamp** attribute is used to identify users who are locked by a reservation timeout.

The relation between **tutor** and **exam** contains two attributes:

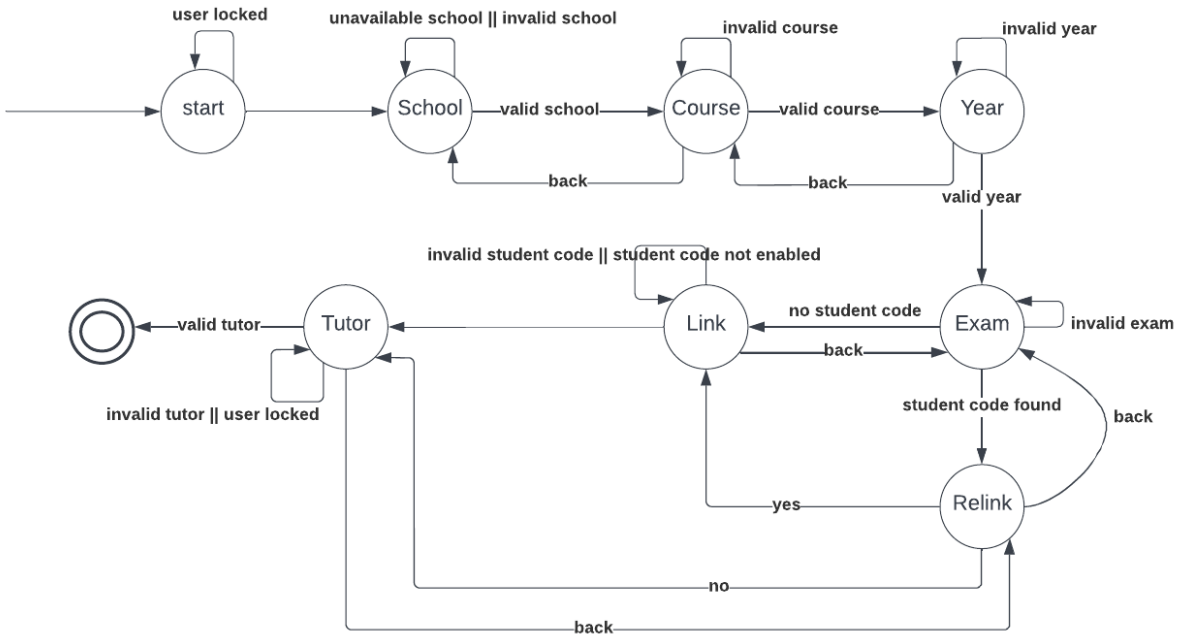
- **available_tutorings:** used to keep track of the amount of available tutorings by a tutor for an exam.
- **last_reservation:** timestamp of the last reservation, used to lock tutor for spam protection.

3. How does the bot work?

A video demonstrating the bot is available at: https://youtu.be/Xnlc61NzWW4

3.1 Bot State machine

The telegram bot works as a state machine and can be easily described as such:



3.1.1 States

Start: the bot awaits any message to start.

School: the bot generates a keyboard containing available schools and sends it to the user.

Course: the bot saves the selected school and sends back a keyboard containing all the courses offered in that school.

Year: the bot saves the selected course and sends back a keyboard containing the available years.

Exam: the bot saves the selected year and sends back a keyboard containing the exams for the chosen School/Course/Year combination.

Relink: the bot saves the selected exam and sends back a keyboard containing the possibility to choose if the user wants to relink his telegram account to a new student code.

Link: if the previous state was **Exam** the bot saves the selected exam, if it was **Relink** the bot deletes the saves userID-personCode association. Then in both cases it sends back a message prompting to insert a student code to link to the users telegram profile.

Tutor: the bot saves the link (also in the DB) and sends back a keyboard containing all available tutors for the chosen exam.

End: the bot reserves the chosen tutoring and locks both tutoring and user.

3.1.2 Conditions

back: the message “indietro” will reset all data from the current state and go to the one preceding it.

user locked: a telegram user is locked from using the bot if he has a active tutoring, or if he tried to reserve a tutor in the last 48 hours *(this time can be changed in the bot config).*

unavailable school: a school is unavailable if it isn't "ICAT" (as requested by the secretariat).

Invalid school: a school is invalid if it isn't present in the DB *school* table.

Invalid course: a course is invalid if it isn't in the chosen school or it isn't present in the DB *course* table.

Invalid year: a year is invalid if it isn't "Y1", "Y2" or "Y3".

Invalid exam: an exam is invalid if it isn't found for the chosen course.

student code found: a student code was found in the DB associated with the telegram user.

invalid student code: student code isn't parsable.

student code not enabled: student code isn't in the *enabled_student* table.

Invalid tutor: tutor with such name wasn't found for the saved exam and course, or tutor hasn't any available tutorings.

3.2 Web API Http Endpoints

All API requests are authenticated. The web server runs on **https** to guarantee the encryption of the Basic Authentication header. The credentials are stored and can be customized from the web config JSON file.

Currently implemented http endpoints:

GET /api/tutoring/
returns all possible tutorings.

GET /api/tutoring/tutor
returns all possible tutorings from a tutor.

GET /api/tutoring/tutor/exam
returns all possible tutorings from a tutor for an exam.

GET /api/reservations/
returns all registered reservations.

GET /api/reservations/reservationID
returns the reservation with the given reservation ID.

GET /api/reservations/processed
returns all processed reservations.

GET /api/reservations/not-processed
returns all non-processed reservations.

PUT /api/tutoring/tutor/exam/student/deactivate
removes an active tutoring.

PUT /api/reservation/id/confirm
confirms a reservation and activates the corresponding tutoring.

4. Indications for use

The code is currently on a public GitHub repository: *https://github.com/zibasPk/PoliTutorBot*

To run the application, valid configuration files are needed, empty templates will be generated at /data/ on the first start up.

4.1 Configuration files attributes

BotConfig

“BotToken”: token needed to communicate with the telegram bot API. This token will be generated while creating a new bot on telegram.

“BotLogLevel”: level at which the global log will be shown. Values are 1 to 6 from highest detail to lowest.

“UserTimeOut”: time in **ms** that need to pass before a conversation is reset for inactivity.

“TutorLockHours”: amount of hours that tutor and user need to be locked after making a reservation.

DbConfig

“Host”: hostname of the MySql database.

“Port”: port of the MySql database.

“User”: username to access the MySql database.

“Password”: password for User to access the MySql database.

“DbName”: name of the database schema.

WebConfig

“Port”: port on which the web API will run.

“WebLogLevel”: level at which the webapp log will be shown. Values are 1 to 6 from highest detail to lowest.

“AuthUsr”: user for http basic authentication.

“AuthPsw”: password for http basic authentication.

5. Conclusion

This Project was completely created with .NET Core and SQL. This has helped me familiarize with C#, a programming language I never used before. I also learned the fundamentals of web API design and to use the Telegram Bot library.
It was my first time working with a client, I learned how to communicate better with non-technical personnel by using sequence diagrams and various types of graphics.

I want to thank the PoliNetwork student association and the secretariat for giving me this opportunity.

Thanks to professor Giovanni Agosta for allowing this to happen and following me in the endeavor.