

Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Computer Programming 4

Talker

Network Text Communicator

author	Amadeusz Drożdż
instructor	mgr inż. Wojciech Dudzik
major	Informatics
year	2019/2020
date	2020-06-17

TABLE OF CONTENTS

1	Project's objective.....	3
2	External specification.....	4
2.1	Application Overview	4
2.2	Client's application architecture	9
2.2.1	Electron.NET and Electron.CLI architecture	10
2.3	Server-side application architecture	11
2.4	Heartbeat client-server communication service.....	12
2.5	Performing of some real usage operations.....	13
2.5.1	User sends a message with an attachment to another user	13
2.5.2	User receives a message with an attachment	14
3	Internal specification	15
3.1	Topics of the thematic tasks.....	16
4	Testing and debugging.....	18

1 PROJECT'S OBJECTIVE

The project's objective was to design and implement a program exploiting object-oriented programming paradigm and its associated tools.

Network user-to-user text communicator was chosen as a topic, as it meets the requirements and provides the possibility of practicing proper internet application architecture creation.

"Talker" is an internet communicator for exchanging text messages between users.

It is similar to a very popular Polish applications, like Gadu-Gadu or messenger Tlen.pl. User can register and login into the system. Every user has its own contacts list. User can also send attachments such as images or video clips and set his status (online, let's talk, do not disturb etc.) and current mood description.

The REST API and the TCP protocol (heartbeat server) are used for transfer messages and attachments. Talker is an client-server application.

2 EXTERNAL SPECIFICATION

2.1 APPLICATION OVERVIEW

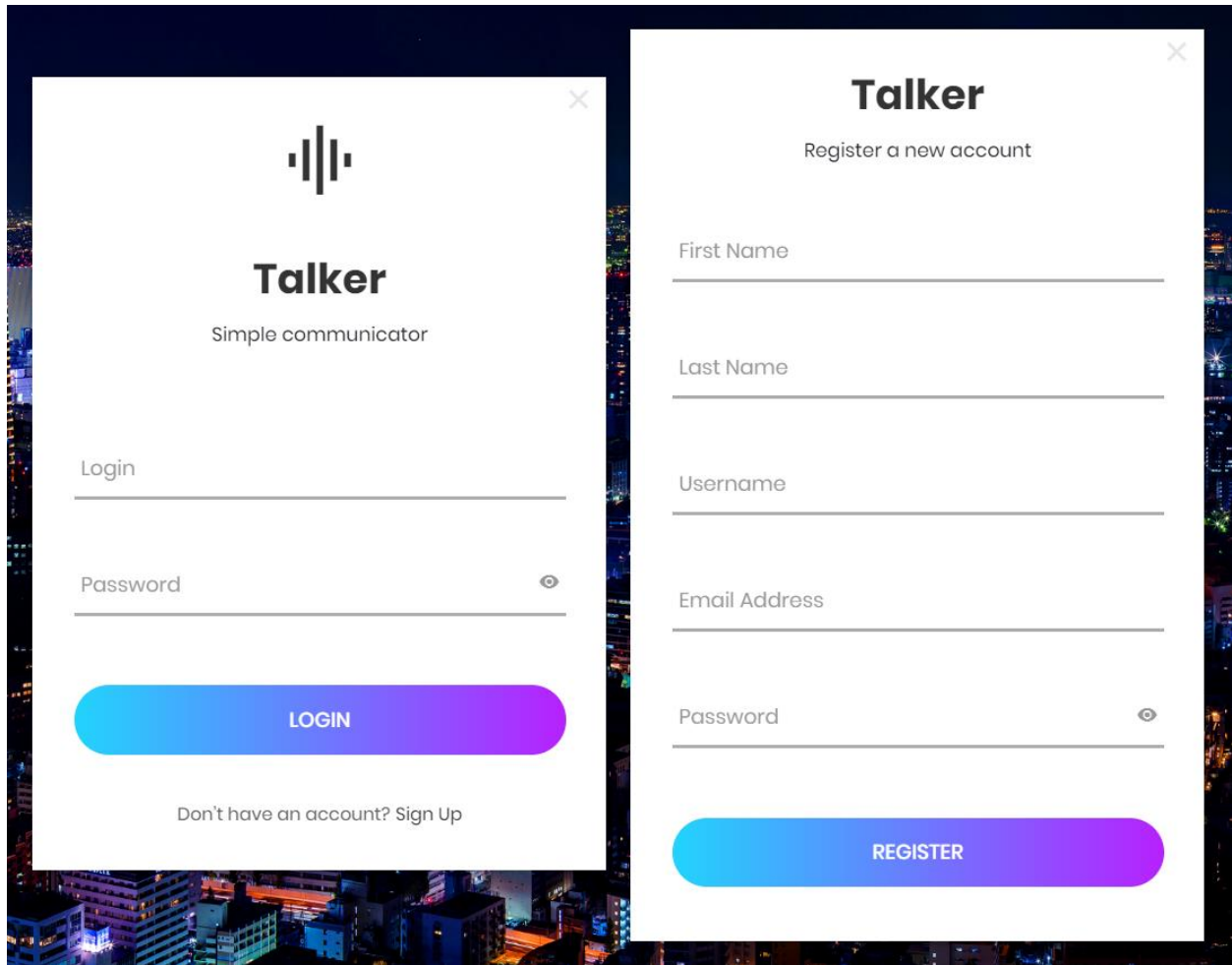


Figure 1: Login and new user registration window.

As I said before, the communicator that I created is very similar to a recently very popular desktop applications, so no one should have any problems with the user interface. After starting the program, login window is displayed in which the user can login into his account. If the user does not have an account yet, a new profile can be created by using option “Sign Up”. New “Register a new account” window will be displayed and after completing, a new account will be created in the system.

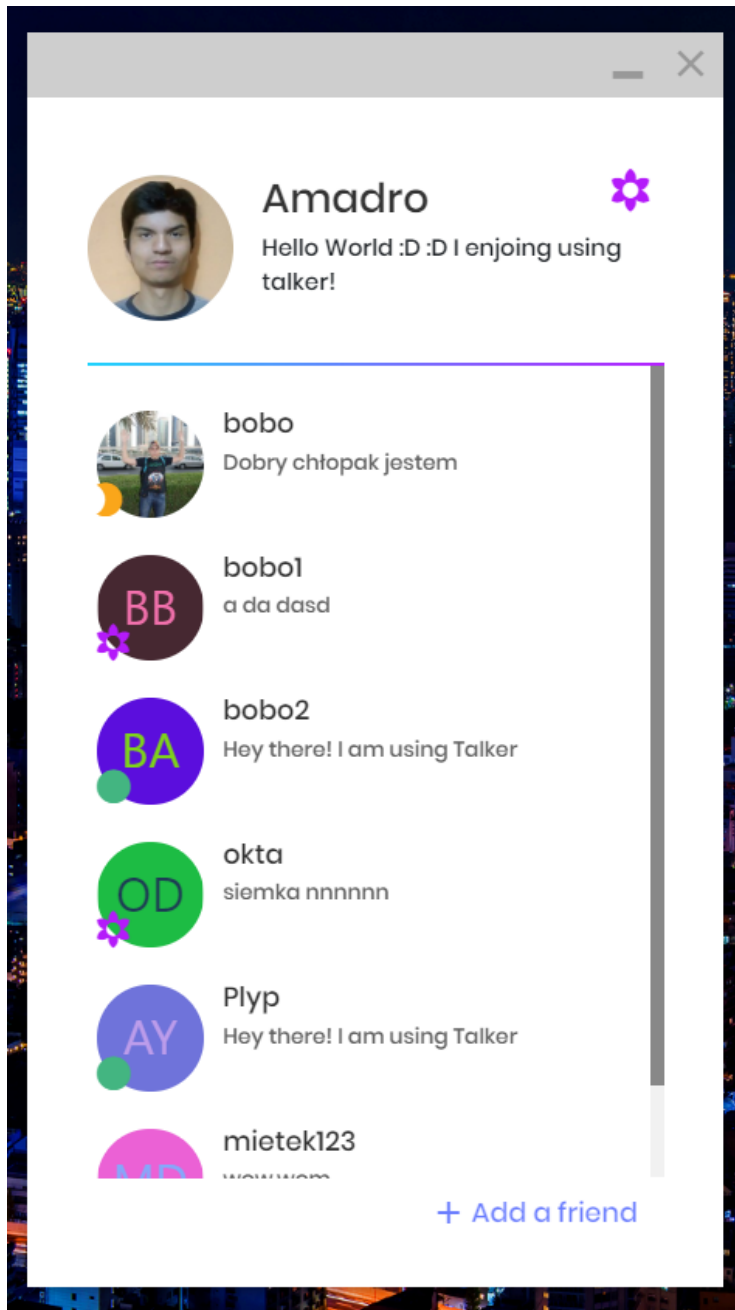


Figure 2: Main window of the program

Main window of the program, which opens up after user login. In the top of section of that window we can change our status and current mood description. Below that, we can see our contacts and manage them, and last but not least, by clicking "Add a friend" we can add a new contact to the list.

Amadro



Hello World :D I love using talker!


Press enter to confirm, unfocus text input to cancel
81 characters left

By clicking on description text we can change it and simply confirm by pressing enter key. Default description is "Hey there! I am using Talker"

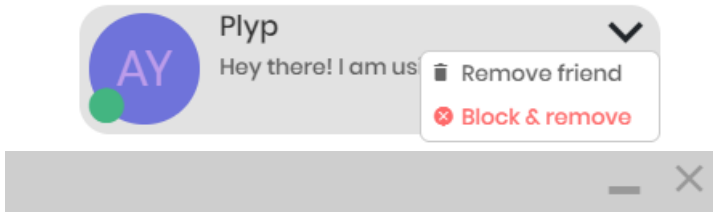
Amadro



Hello World talker!

- ☒ Online
-  Let's talk
- ☐ Idle
- ☐ Do Not Disturb
- ☐ Invisible

We can change our status by hovering over on a currently chosen one and pick the new one.



Contacts can be managed by hovering over selected contact and clicking on the arrow button.

Block a friend

Are you sure you want to block Plyp?

⚠ Blocked user will not be able to send you messages. If you want to unblock a user, add them to your friends list

Block user

Cancel

We can remove contact from friends list or block all communication (ban) with selected contact.

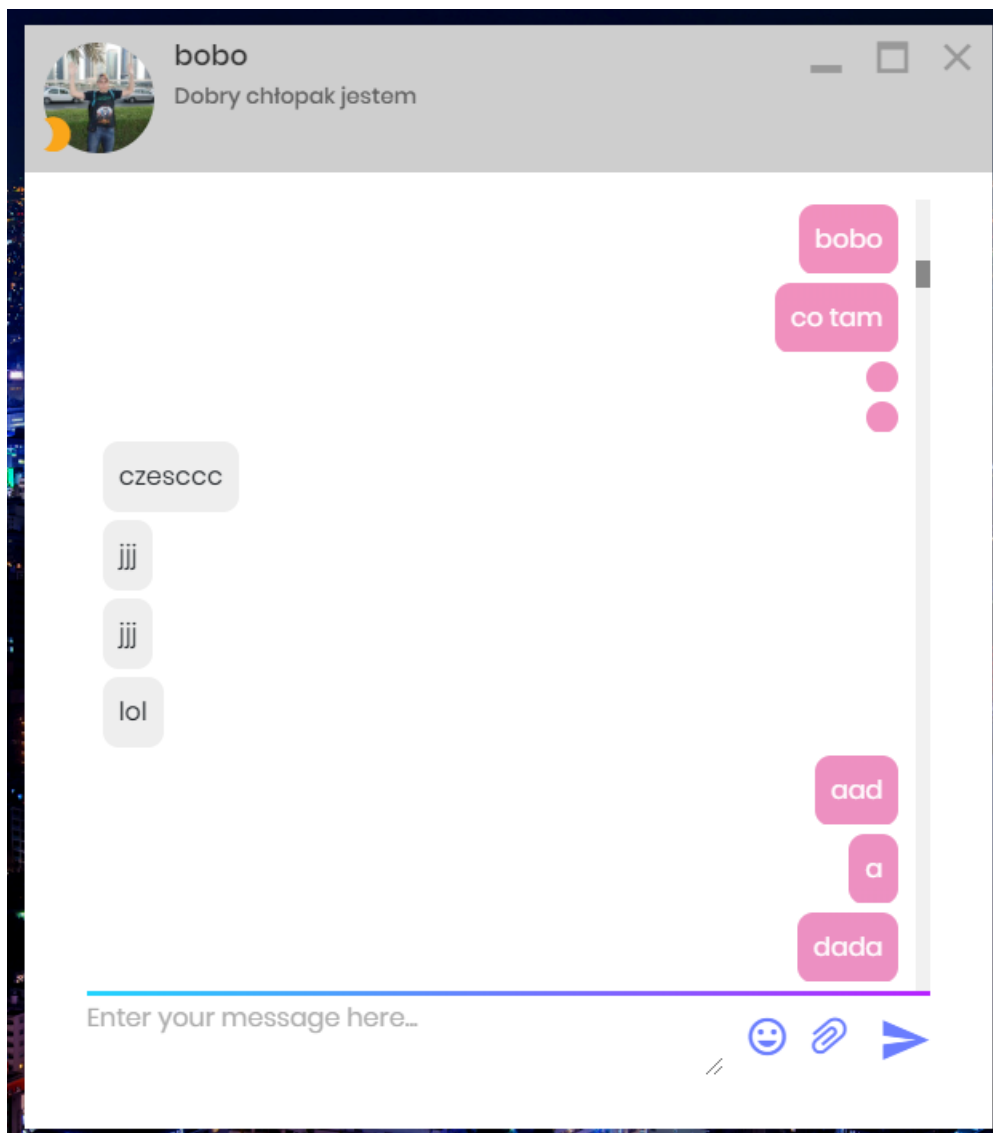


Figure 3: Chat window with selected user

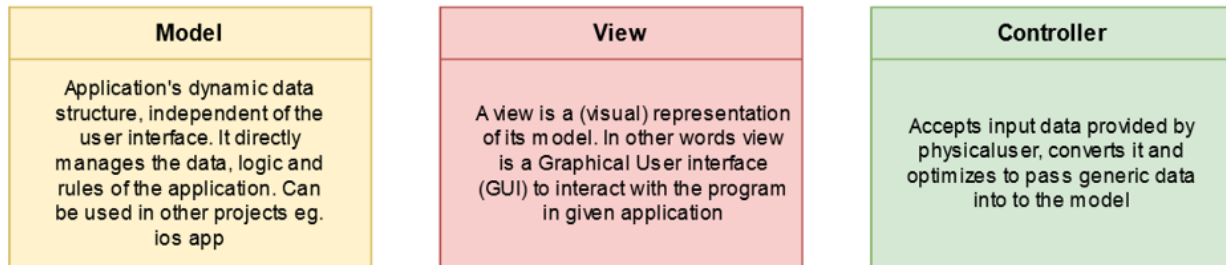


Figure 4: Attachment sending panel.

Chat window where we can send text messages or attachments for example image or movie clip to selected user.

2.2 CLIENT'S APPLICATION ARCHITECTURE

In client application I used Model-View-Controller a design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements.

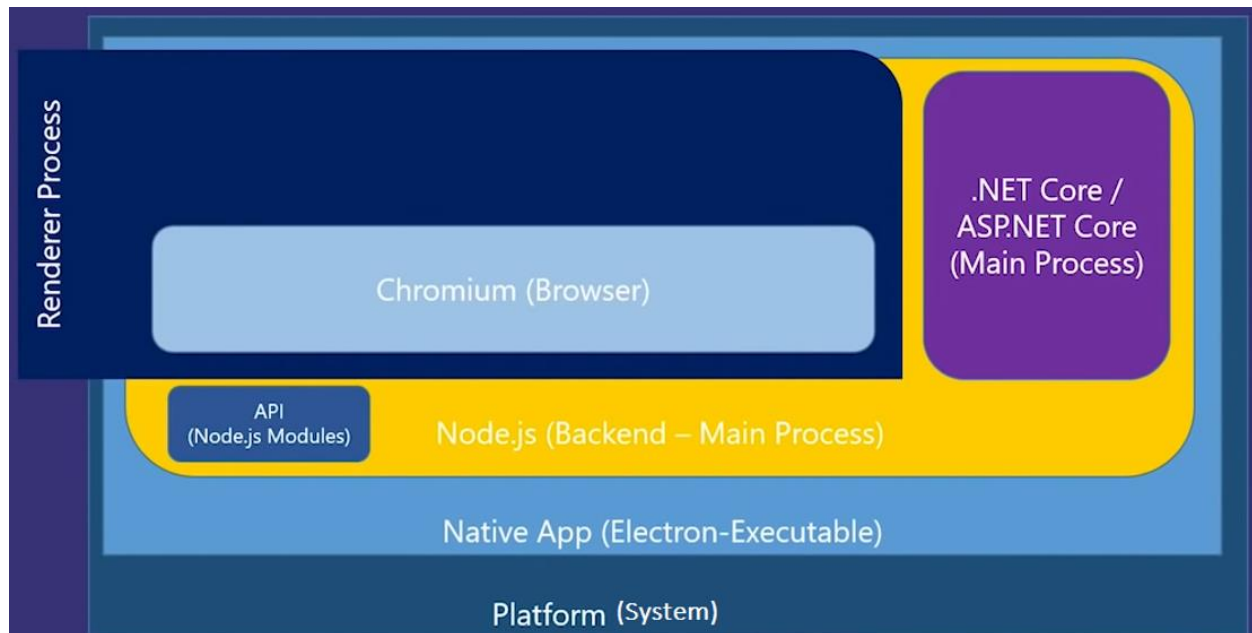


In view I used Electron.NET framework (wrapper to electron.JS) and Node.JS to communicate with NET Core Application (controller of the application). In GUI logic (operations such as client-side form checking or some popup dialogs etc.) I used JQuery framework. To create Layout I decided to use plain HTML 5 and CSS 3 to create an interface as light as possible without any unnecessary code and heavy frameworks

In controller I also used Electron.NET (Electron NET API) to communicate with View.

In the model I used RestSharp library to send API requests which is very stable and easy to use c# http client.

2.2.1 ELECTRON.NET AND ELECTRON.CLI ARCHITECTURE

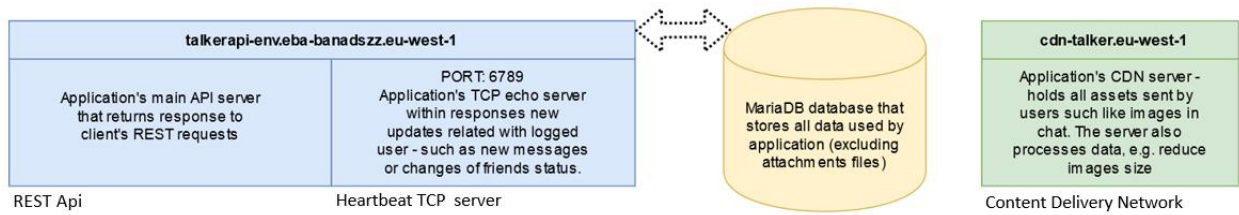


Electron.NET architecture is almost the same as Electron.JS which is Node.JS module. But Electron.NET is a wrapper around a "normal" Electron application with an embedded ASP.NET Core application to run node.js modules.

Via Electron.NET IPC bridge we can invoke Electron APIs from .NET code.

The Electron.CLI extensions provide toolset to build, start and debug Electron.NET applications.

2.3 SERVER-SIDE APPLICATION ARCHITECTURE



All server side applications are deployed on Amazon AWS Elastic Beanstalk platform. For API and TCP heartbeat echo server I used Spring Boot, JPA, Hibernate and Spring Integration, which all are java severing technologies. To store data I used MariaDB database (refreshed version modernized version of MySQL server). And finally on CDN server I used PHP 7 and FineUploader framework. The Content Delivery Network server - holds all assets sent by users such as images or media clips in chat. The server also can process some data, for example by using FineUploader features server can reduce images size by shrinking it.

List of API Endpoints:

`auth/login`

`auth/register`

`/api/contacts/list`

`/api/contacts/remove`

`/api/contacts/add`

`/api/messsages/send`

`/api/messsages/list`

`/api/user/me`

`/api/user/update`

`/api/contacts/block`

2.4 HEARTBEAT CLIENT-SERVER COMMUNICATION SERVICE

SocketChangesUpdater is a part of clients-server communication service. The client's application listens for changes that are related with the user using tcp connection - such as new messages or changes of friends status. These messages don't need to be encrypted because they don't contain any crucial data - they are only used as a trigger for encrypted REST API request. eg. pull new messages from user id 5.

For example: our user id is equal to 4 the request to TCP heartbeat server will look like this:

"4; " were "4" is user id.

If we have incoming messages from for example user id 2 and user id 7 echo response will be as follows:

"1;-2-7-" were "1" means that we have new incoming message and -2-7- indicates sender users ids.

But If we did not receive any new updates, response will be simply ";0;"

Heartbeat status requests are sent every 3 seconds.

Possible heartbeat responses:

```
UPDATE_TOO_SOON( value: -1),  
NOTHING_NEW( value: 0),  
NEW_MESSAGE( value: 1),  
FRIENDS_UPDATE( value: 2),  
NEW_MESSAGE_AND_FRIENDS_UPDATE( value: 3),  
NEW_FRIEND( value: 4),  
FRIEND_REMOVED( value: 5),  
FIRST_UPDATE( value: 6);
```

2.5 PERFORMING OF SOME REAL USAGE OPERATIONS.

2.5.1 USER SENDS A MESSAGE WITH AN ATTACHMENT TO ANOTHER USER

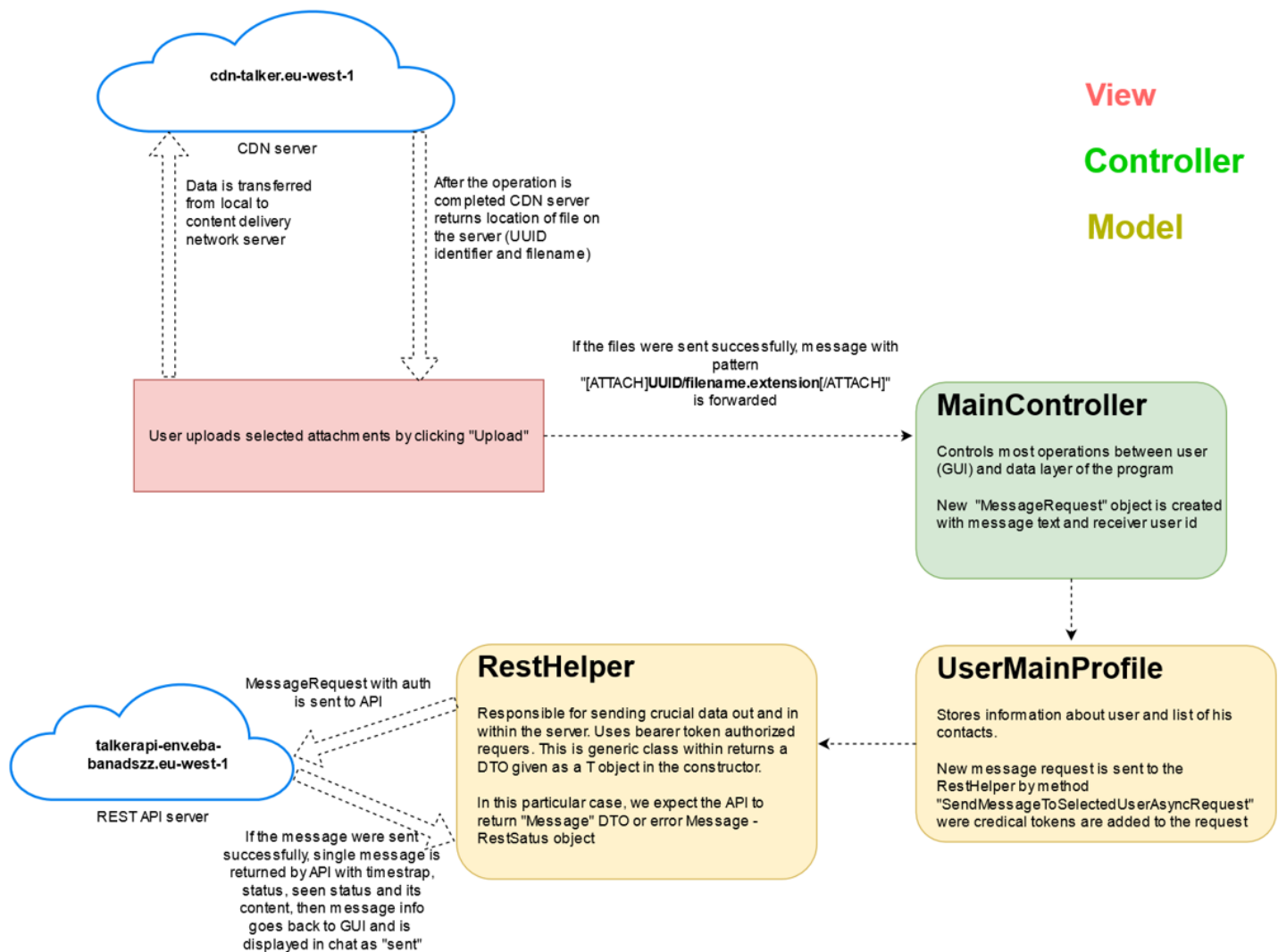


Figure 5: The figure shows performing an operation of sending a message with an attachment to another user

2.5.2 USER RECEIVES A MESSAGE WITH AN ATTACHMENT

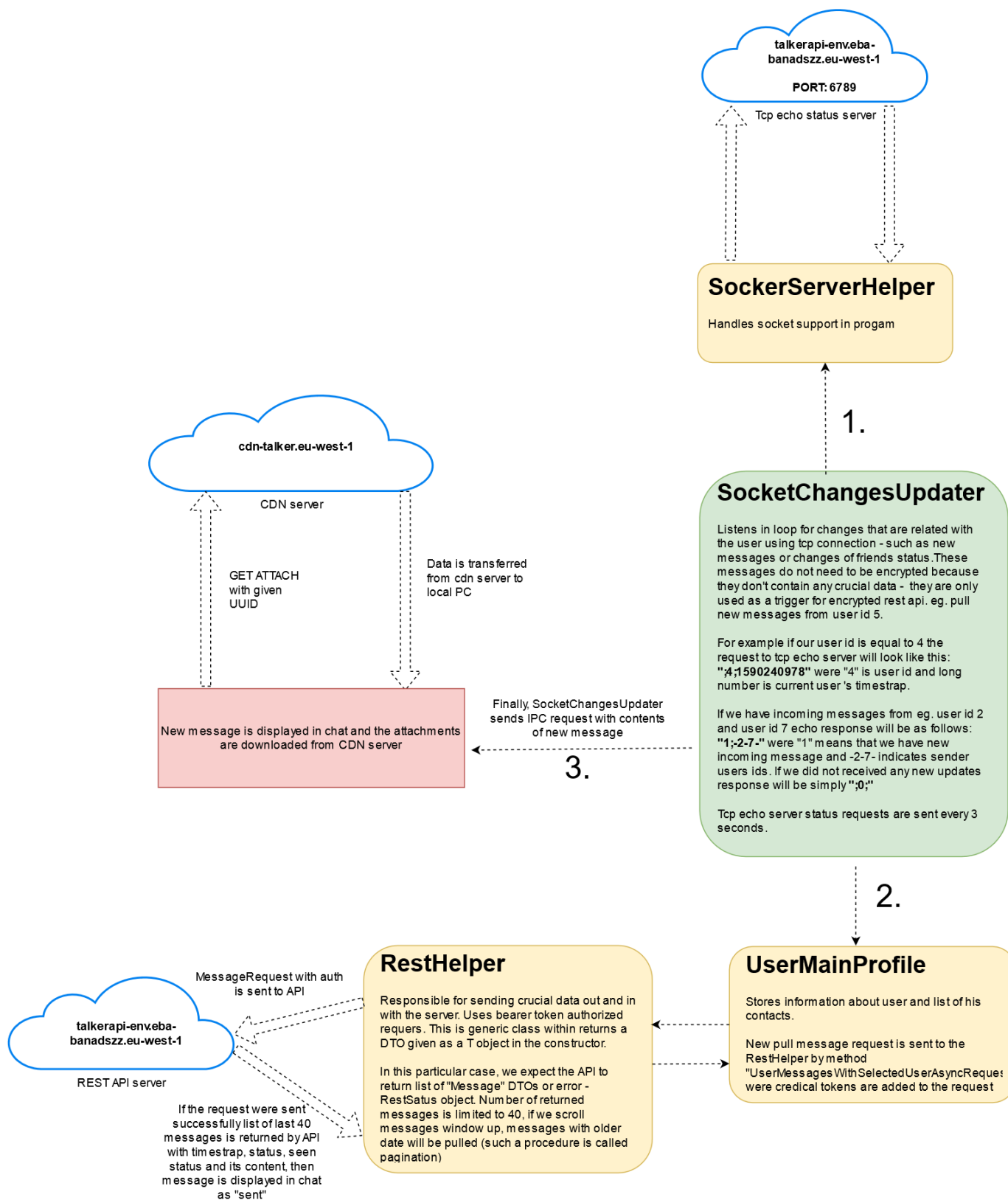


Figure 6: The figure shows performing an operation of receiving a message with an attachment

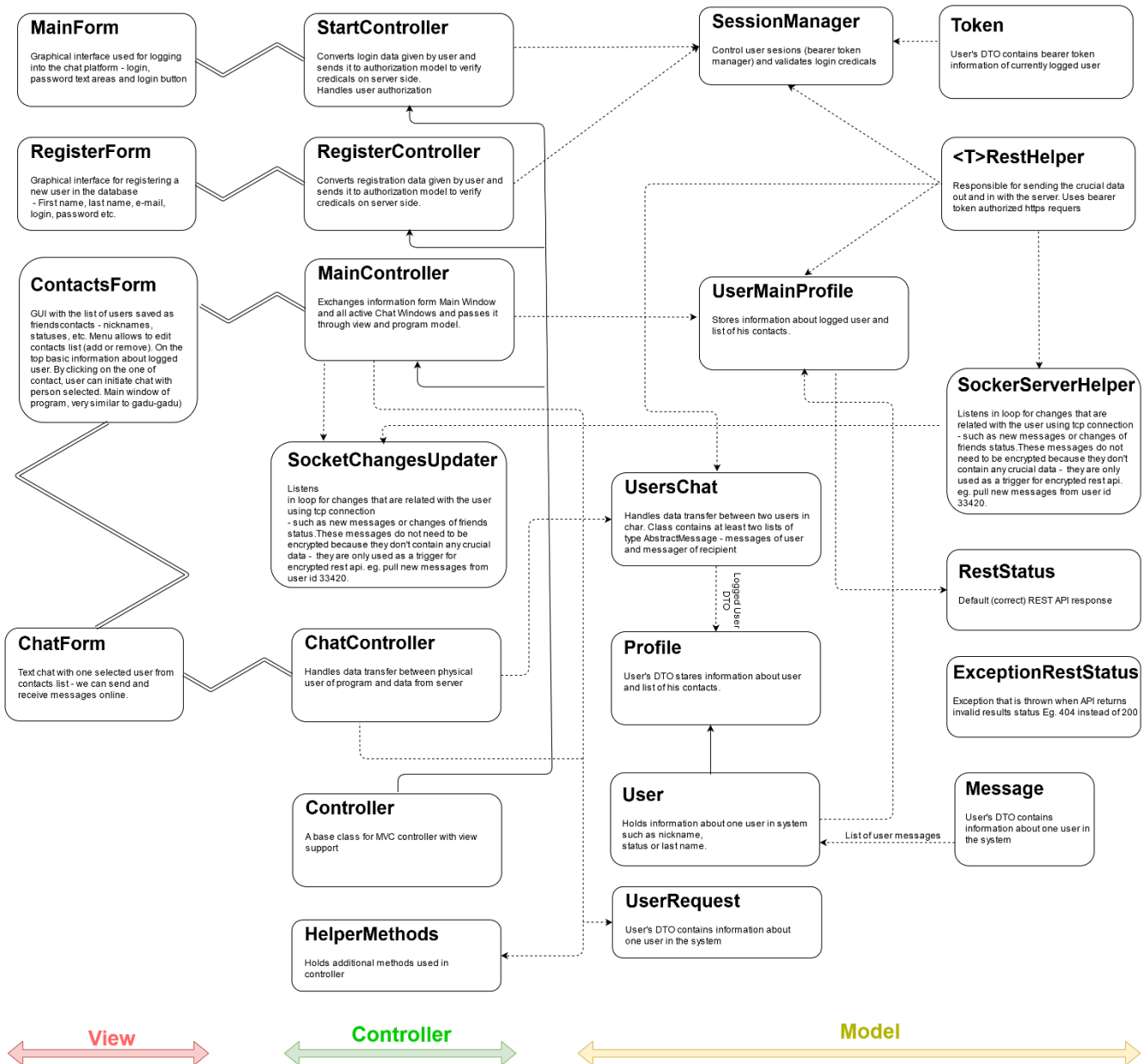
3 INTERNAL SPECIFICATION

Simplified graphical class diagram

Graphical User interface (GUI)
to interact with the program in
specific application - in this case
made with electron.NET

**Accepts input data provided by physical
user, converts it and optimizes to pass
generic data into to the model**

**Application's dynamic data structure, independent
of the user interface. It directly manages the data,
logic and rules of the application. Can be used in
other projects eg. ios app**



Amadeusz Drożdż, student 2 roku informatyki po angielsku

Figure 7: Client's app diagram

The diagram:

Dotted lines -> marks references to class objects or static classes;

Continuous line -> inheritance

Double lines -> marks the IPC communication between JS and C#.

Due to the fact that the image may be illegible it's recommend do download image in full size.

Class diagram in full size: <https://imgur.com/a/GKzFEvS>

More specific description of types and classes is moved to the appendix: doc folder on project's github.

3.1 TOPICS OF THE TEMATIC TASKS

The project meets at least 5 of 8 thematic tasks.

List of completed thematic tasks:

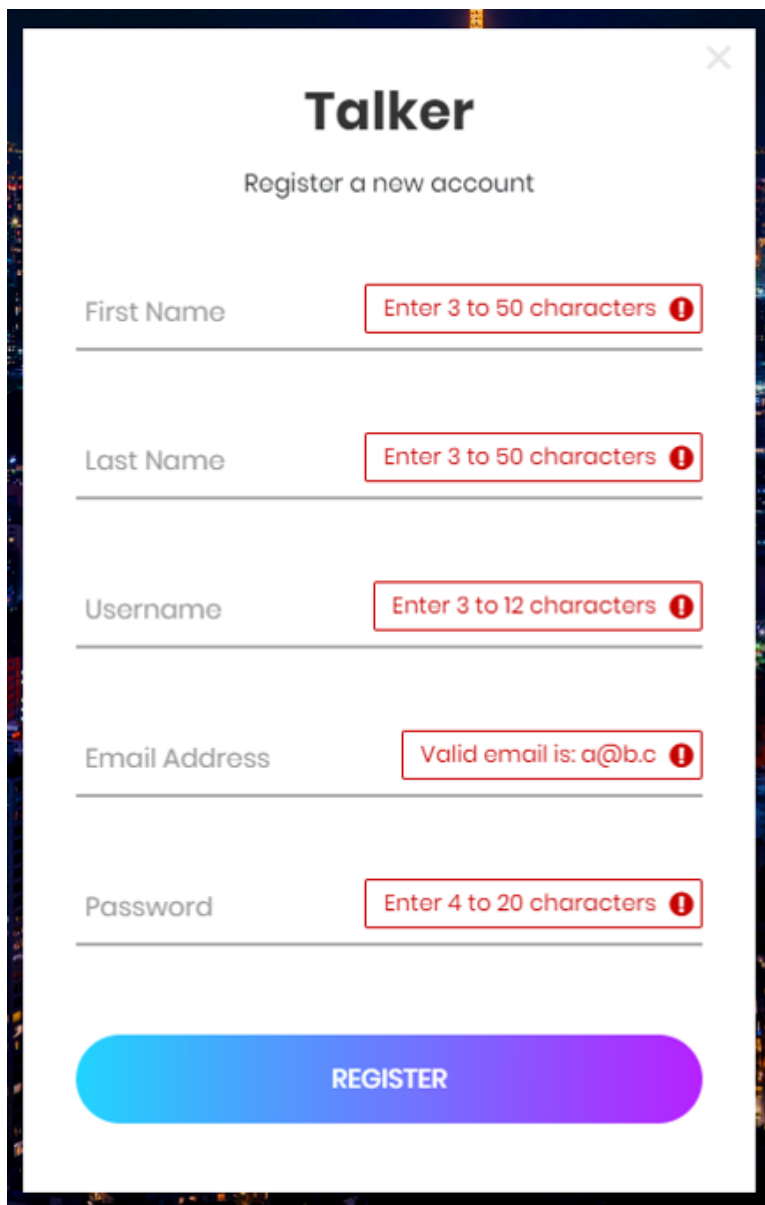
- Run-time type information:
The data type at runtime mechanism has been used many times in this project. For example, json objects are converted in real time to its DTO's, additionally if a server-side error occurs API response (with returned error description) is automatically converted to appropriate "ExceptionRestStatus".
- Exceptions
Due to the fact that the project uses Electron.JS technology it's very important to catch errors and transfer it to corresponding page in view – display error message to user. Program does not show the console and does not use the WinAPI technology so any uncaught errors may be missed or in the worst case they can cause the application to hang without any GUI callbacks.
- Templates
On the client side, a generic class (RestHelper) is used many times to automatically convert the Rest API returned JSON object to its DTO's.

➤ **Threads**

In the client's application to prevent GUI freezing almost every back-end operation is started in new thread (c#'s async await mechanism is used) and has it's time limit. For example, for API REST calls time limit is 2s. The heartbeat tcp requests are sent in a separate thread (every 3 seconds to check's new client updates).

➤ **Regular expressions**

In almost every network application it is necessary to use the regex mechanism. In the application, the data sent by the user are validated by corresponding regex pattern. For example during registration the integrity of the e-mail is checked.



The image shows a registration form titled "Talker" with the subtitle "Register a new account". The form contains five input fields, each with a red validation message box to its right:

- First Name**: "Enter 3 to 50 characters" with an information icon.
- Last Name**: "Enter 3 to 50 characters" with an information icon.
- Username**: "Enter 3 to 12 characters" with an information icon.
- Email Address**: "Valid email is: a@b.c" with an information icon.
- Password**: "Enter 4 to 20 characters" with an information icon.

At the bottom of the form is a large, rounded button with a blue-to-purple gradient, labeled "REGISTER". The entire form is enclosed in a white box with a close button (X) in the top right corner.

4 TESTING AND DEBUGGING

While creating this project I came across many problems, but luckily I was able to solve all of them. Main problems I found during writing this project were strictly related with GUI technology that I used – Electron.NET. I had trouble with “disconnecting” view (html-js page) from it’s controller - I was able to solve this bug by dividing the program into threads more efficiently. I also had an issue with loading assets form Content Delivery Network server – it turned out that the problem lies within the Electron.NET side. So I made an issue on Electron.NET project’s github ([link to my github issue](#)) and after contacting to the creator we managed to solve the problem. The final program has been tested many times and it’s very stable. I learned a lot of new things by creating this project. My abilities have improved quite significantly in terms of making vast client-server applications.