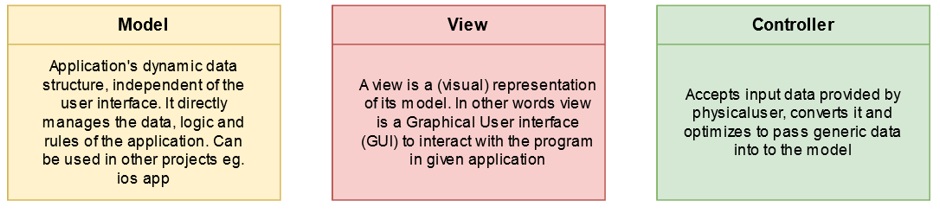
I created a simple application for exchanging messages between users. In this application you can also send attachments such as images or video clips. User can also set his status (online, let’s talk, do not disturb etc.) and current mood description.

This is login and new users registration window.

And here on the right we have main window of the program which opens after user logsin, we can change our status and description. Below that we can see all our contacts and manage them, and last but not least by cliking “Add a friend” we can add a new contact to the list. On the left side you can see chat window with selected user. Here we can text that user or we can send them some attachments for example image or movie clip. I will show you how it works in real life after the presentation.

**Client's application architecture**

In my 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.

****

So. basically Users puts data in View then the data is converted, optimized and validated in Controller. And the generic optimized data is passed to Model.

A few words about technologies that I used in client’s application.

Well maybe I will start in view. So in view I used Electron.NET framework (which by the way is a wrapper to electron js) and Node.JS to communicate with net core app (I mean by that controller). In GUI logic (so operations such as client-side form checking or some popup dialogs esetra) I used JQuery framework. To create Layout I decided to use plain html5 and css3 to create an interface as light as possible without any unnecessary code and heavy frameworks

In controller I also used Electron.NET (strictly speaking Electron NET API) to communicate with View. I will say a few more words about Electron NET in further slides.

In model I used RestSharp library to send API requests which is in my opinion very nice and easy to use http client.

**Electron.NET and Electron.CLI architecture**So basically 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 provides toolset to build, start and debug Electron.NET applications.

**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. And 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 FINEULOPADER features server can reduce images size by shrinking it.

**Presentation of some application possibilities**

Soo. Ok now I wanted to show you some scenarios when client preforms some operation on the application.

**User sends a message with an attachment to another user**

So In this case we are starting in red rectangle which is GUI – user starts sending a new attachment by cliking “upload” on chat form. After that all selected data files by user are transferred from local PC to the content delivery network server. After all the files were uploaded successfully without any errors, he CDN server return location of files on server. Then through elektron IPC message with pattern like here ([ATTACH]….[/ATTACH]) is forwarded to MainController.

MainController crates a new object of class “MessageRequest” with message text and receiver user id

Next

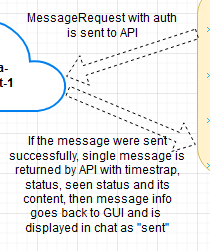
newly created “MessageRequest” object is send to RestHelper by method "SendMessageToSelectedUserAsyncRequest" were auth tokens are added to the request/

MessageRequest which credicals is sent to RestHelper.

**RestHelper**

Responsible for sending crucial data out and in within the server. Uses bearer token authorized requests. This is generic class within returns a DTO given as a T object in the constructor.

In this particular case, we expect the API to return "Message" DTO or error status - RestSatus object



**User receives a message with an attachment**

**Well in this case our journey begins in SocketChangesUpdater**which is a TCP Echo client

Witch listens in loop 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.

Ex2. if our user id is equal to 4 the request to tcp heartbeat server will look like this:

";4;1590240978" were "4" is user id and long number is current user 's current timestrap.

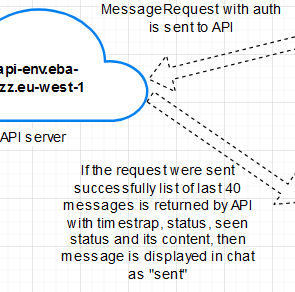
If we have incoming messages from for example user id 2 and user id 7 echo response will be as follows:

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

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

Heartbeat status requests are sent every 3 seconds.

So moving next…. in our scenario we have situation where we have received a new message. In this case appropriate class and proper is method needs to be called.  
  
In the in case of pulling new messages , method "UserMessagesWithSelectedUserAsyncRequest" form “UserMainProfile” is used, were credical tokens are added to the request and it is sent to RestHelper class.

In this particular case, we expect the API to return list of "Message" DTOs or error - RestSatus error object. Number of returned messages is limited to 40, if we scroll messages window up, messages with older date will be pulled (such a procedure is called pagination)

**Full diagram – client’s app**

This is client’s side application diagram, but it has not been updated lastly, so not everything is up-to-date There is DTOs missing on diagram.