Real-time visualization of analyzed industrial communication network traffic

PSE Report of

Xiaoru Li, Klevia Ulqinaku, Mario Alberto Gonzalez Ordiano, Philipp Mergenthaler

KOMPETENZZENTRUM FÜR ANGEWANDTE SICHERHEITSTECHNOLOGIE
Advisor: M.Sc. Ankush Meshram

This Document describes the development process of the Adininspector visualization tool.

Version 1.0.0

Contents

1 Introduction

The Adininspector visualization tool was developed using the waterfall software development process. This document contains the reports for the individual process phases.

2 Requirements

2.1 Motivation

The goal of this project is to create a software visualization tool for industrial network traffic to simplify the analysis of anomalous behaviour, both in realtime and from stored captured data.

This software is part of the Anomaly Detection in Industrial Networks (ADIN) framework and is referred to as the "ADIN Inspector". The ADIN framework serves to first capture network traffic in industrial networks. This traffic data is then dissect and analyzed for anomalies using machine learning. A streaming platform is used to provide access to both network data and the notifications produced by the dissectors. Finally, a GUI component visualizes the network traffic and anomalies. One component to achieve this goal is a web interface built with modularity in mind so as to make it easily extendable.

The Web view is able to display a series of diagrams and charts to easily identify the behaviour of the network. Within the Web view the user has the ability to zoom, select, highlight, and filter out data to better understand the aforementioned behaviour at different OSI layers, as well as visualize the flow rate between network nodes.

To support this Web view a back-end messaging solution is needed. This allows the user to easily switch between multiple streams of captured data.

2.2 Overview

While computers can analyze raw data easily, for humans it's easier to recognize behavior using visual aids. Visual analytics is utilizing graphics to enhance human cognition to understand a problem better or find 'a needle in the haystack' within a huge amount of data. Making it an invaluable tool for security analysis.

Industrial network security aims to understand the communication network traffic generated in an industrial production system. Analyzing the traffic generated by underlying industrial protocols is the primary step.

Real-time visualization of analyzed network data will help the end-user to understand the system's communication behavior and changes within it more clearly. Deviations or incidents can be detected visually by operators as they are occurring or already persisting.

Supplemental to this, the ability of operators to run an analysis of past anomalous behavior can help strengthen infrastructure for future failure or attacks. As well as provide useful information for training of new security analysts.

2.3 Interfaces

The environment of the project is a modern browser with internet access. The project is OS-agnostic. Therefore the underlying Operating system is irrelevant on the user's end.

2.3.1 Software

- Client
  - web-browser of the latest generation.
- Server:
  - Java
  - Kafka messaging framework
  - MongoDB database

  Kafka and OpenJDK, a Java implementation, are open source products and MongoDB is released under a license that is free for non-public use.

2.3.2 Hardware

- Client:

  System capable of network connectivity
- Server:
  - Network capable system
  - System capable of running all backend-software components
  - System with adequate storage

2.4 Functional Requirements

2.4.1 Must Have

FR100 When opening the Web view the user has to be greeted by a login screen.

FR110 The user has to be able to log out of the system.

FR200 There need to be at least two levels of access for different account types (aka security roles). Level of access is defined as: the specific set of data streams the user is able to view and select to analyze.

FR300 Once logged in, an user has to be able to select a data stream to be visualized.

FR400 The user has to have the ability to select multiple diagrams to visualize the selected data stream.

FR500 The user can use at least these diagram types:

- A timeline plot
- A scatter plot
- A Network diagram

FR600 The user is able to dynamically change which components of the data are used for the X and Y axis of the diagram.

FR700 The user should be able to add new diagrams to the GUI and configure them (i.e. setting diagram type and axis) both at creation and at a later time.

FR710 The GUI has to support a minimum of 4 different diagrams at once.

FR720 Each diagram should be able to be maximised to take on the full size of the diagram container.

FR800 Each drawn diagram can be connected to a different data stream.

FR900 The amount of data can be limited via a slider, effectively setting a limited time window, to which all diagrams must update to.

FR910 Within the slider the user is able to scroll through the timeline and the diagrams need to react in real-time.

FR1000 There needs to be an auto scroll function (play button) which automatically scrolls through the selected time window and whose speed is adjustable.

FR1100 ADIN Inspector has to have a function to pick any data point and show all its information.

FR1110 ADIN Inspector has to support, for node-link diagrams, both picking of nodes and links

FR1200 ADIN Inspector has to offer a function to select one or more data points.

FR1210 ADIN Inspector has to offer a function to create a new diagram showing the selected data points.

FR1300 ADIN Inspector has to offer a filter mechanism for global filters and per-diagram filters.

FR1310 The user should be able to enable one or more global filters and disable them again.

FR1330 ADIN Inspector should provide at least these filter types:

- Passing only packets with a specific value for a specific key
- Filtering according to one of these comparison operators: <, <=, >, >=
- Computing a moving average

2.4.2 Should Have

FR1320 The user should be able to enable one or more per-diagram filters for each diagram and disable them again.

FR1332 ADIN Inspector should provide this filter type:

Computing a data rate or flow rate

FR1400 The GUI should be able to support an undeterminate number of diagrams and scrollbar.

2.4.3 Nice To Have

FR590 Extended selection of diagram types (e.g. temporal raster plot and scatterplot matrix).

FR1390 The GUI should have functionality to interactively change filter settings (Sliders, drop down menus etc.)

FR1500 Save the session state and display it in the user profile.

2.5 Data Requirements

2.5.1 User Data

(DR100)

- User ID *(Unique)*
- User Name *(Unique)*
- User Password
- User role

2.5.2 Event stream data for Web UI

(DR200)

- Timestamp
- 2 types of events for both notifications (eg. warning or errors) and data points
  - Notification
    - Severity level
    - Notification content
  - Data Point
    - Packet type, eg. protocol or network layer
    - Packet summary (incl. packet length)

* Source
* Destination

2.5.3 Raw packet metadata

(DR300)

Raw packet metadata include Time, Source, Destination, Protocol and Packet Length.

2.6 Non-Functional Requirements

NF100 The rendering latency should be no longer than 2 seconds.

NF200 The web UI should be viewable on modern web browsers.

NF300 The web UI should not crash when network connection is unstable.

NF310 The system should not crash when malformed or incomplete data are received.

NF320 The system should be able to be recovered with ease from a crash.

NF400 The framework should be able to handle data streams from at least 100 physical nodes in the network.

NF500 The system should have a mechanism in place that is able to deny access from unauthorized personnel.

NF600 The data visualization should be easily understandable or learnable for non-professionals.

NF700 The web UI should be accessible to all user groups, including people with conditions like color blindness.

NF800 The web UI should be easily extendable with new diagram types.

NF810 The web UI should be easily extendable with new filter and aggregation components.

NF820 The web UI should be easily extendable with new input data types.

2.6.1 Quality Requirements

Product Quality	really good	good	normal	not relevant
Functionality				
Accuracy	x			
Interoperability			x	
Security	x			
Reliability				
Error tolerance			x	
Recoverability			x	
Usability				
Understandability		x		
Learnability				x
Usability	x			
Efficiency				
Time behaviour		x		
Consumption behaviour		x		

2.7 Essential Test cases

T100 Successful login

Precondition Open browser window.

Action The user enters the URL for the ADIN web server in the address bar and presses Enter.

Reaction The browser loads the ADIN website and shows the login screen.

Precondition ADIN website had been opened and show the login screen.

Action The user enters a valid username and the corresponding password.

Reaction Successfully logged in. The browser loads the ADIN main screen with one empty diagram.

T101 Failed login (wrong username or wrong password)

Precondition Open browser window.

Action The user enters the URL for the ADIN web server in the address bar and presses Enter.

Reaction The browser loads the ADIN website and shows the login screen.

Precondition ADIN website has been opened and shows the login screen.

Action The user enters either a valid username and an incorrect password or a non existing username and an arbitrary password.

Reaction The login screen shows an error message that the username or password is incorrect and the entry field for the password is cleared.

T200 Open second diagram.

Precondition The browser has been logged in to ADIN and shows the main screen with one empty diagram.

Action The user presses the "New Diagram" button on the top right side of the screen.

Reaction The browser opens a modal with diagram settings

Action The user presses the create button at the bottom left

Reaction The browser opens a second diagram, splitting the diagram panel in to two.

T210 Open multiple diagrams

Precondition The browser has been logged in to ADIN and [T200] has been passed.

Action The user repeats [T200] two more times.

Reaction The diagrams grid now displays four diagrams in a 2 x 2 formation.

T220 Configure a diagram. (Concrete case: timeline diagram of packet size)

Precondition The browser has been logged in to ADIN and shows the main screen with one empty diagram.

Action The user selects "Timeline Diagram" from the Diagram selection box.

Reaction The diagram changes to a timeline diagram. The x-axis is labeled with "time [s]".

Action The user selects "Packet size" in the "Y-Axis" selection box.

Reaction The diagram's y-axis is labeled with "size [bytes]".

T300 Filtering

Precondition The browser has been logged in to ADIN and shows at least one diagram.

Action The user enables a filter from the global filters section.

Reaction The diagram now only shows filtered data

Action The user disables the same filter.

Reaction The diagram shows original data again

T310 Filter chaining

Precondition Logged in to ADIN and at least one diagram and one global filter is active.

Action The user enables another filter from the global filters section.

Reaction The diagram now only shows relevant data

T400 Full screen a diagram

Precondition Logged in to ADIN and shows at least two diagrams

Action The user presses the full screen button on the top right corner of the diagram

Reaction The diagram's window is maximized to the diagram container of the web page.

T450 Full screen and exit Full screen

Precondition Logged in to ADIN and shows at least two diagrams

Action The user presses the full screen button on the top right corner of the diagram

Reaction The diagrams window is maximized to the diagram grid of the web page.

Action The user presses the exit full screen button on the top right corner of the diagram

Reaction The diagram grid of the web page is restored.

T500 Play button basic functionality

Precondition Logged in to ADIN and shows at exactly one diagram.

Action The user presses the play button at the bottom left of the web page.

Reaction The play button turns into a stop button.

Reaction The diagram updates according to the time shown at the play head. This means the diagram displays data from the start time of the data unitl the time shown at the play head

Action The user presses the stop button.

Reaction The diagram remains static with the currently displayed data.

T510 Time window(s)

Precondition Logged in to ADIN with one diagram. The play head is in its default position all the way to the left

Action User drags the left diagram time window key-frame to the right.

Reaction The timestamp above the keyframe updates according to it's position

Action User drags the right diagram time window key-frame to the right.

Reaction The timestamp above the keyframe updates according to it's position

T600 Investigate a data point

Precondition A diagram showing data points.

Action The user hovers with the cursor over a data point.

Reaction ADIN Inspector opens a tooltip which shows all values or key-value pairs of this data point.

T610 Investigate nodes and links in a node-link diagram

Precondition A node-link diagram showing data points.

Action The user hovers with the cursor over a node.

Reaction ADIN Inspector opens a tooltip which shows information about this node.

Action The user hovers with the cursor over a node.

Reaction ADIN Inspector opens a tooltip which shows all values or key-value pairs of the data point that this line represents.

T620 Select data points

Precondition A diagram showing data points.

Action The user clicks on a data point.

Reaction ADIN Inspector marks this data point, e.g. by highlighting.

Action The user clicks on a different data point.

Reaction ADIN Inspector marks this second data point, e.g. by highlighting, and unmarks the first data point.

T630 Create a new diagram based on a selection, where the new diagram has a different type than the first diagram

Precondition A diagram showing data points.

Action The user selects one or more data points.
The user right-clicks the selection.
In the pop-up menu the user selects "create new diagram from selection"

Reaction ADIN Inspector opens a new diagram.

Action The user selects a diagram type.

Reaction ADIN Inspector displays the selected data points in the new diagram.

2.8 Software Modeling

2.8.1 GUI

In this subsection we'll discuss the GUI, it's components and the workflow of a typical user.

2.8.1.1 Main Window

As the programm is opened the user is greeted with a login screen shown in Figure 2.1. After the user has entered it's credentials and clicked on the login button he is shown Figure 2.2.

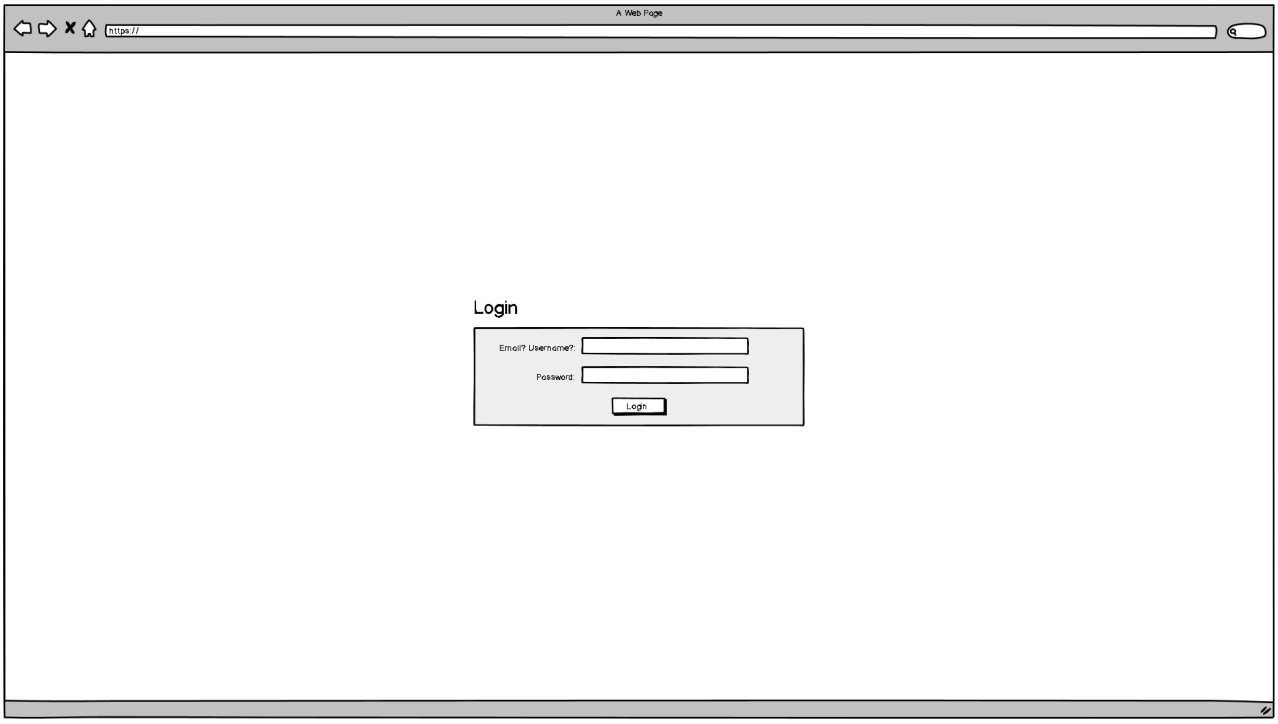


Figure 2.1: The Login Windows shown as the page is first loaded

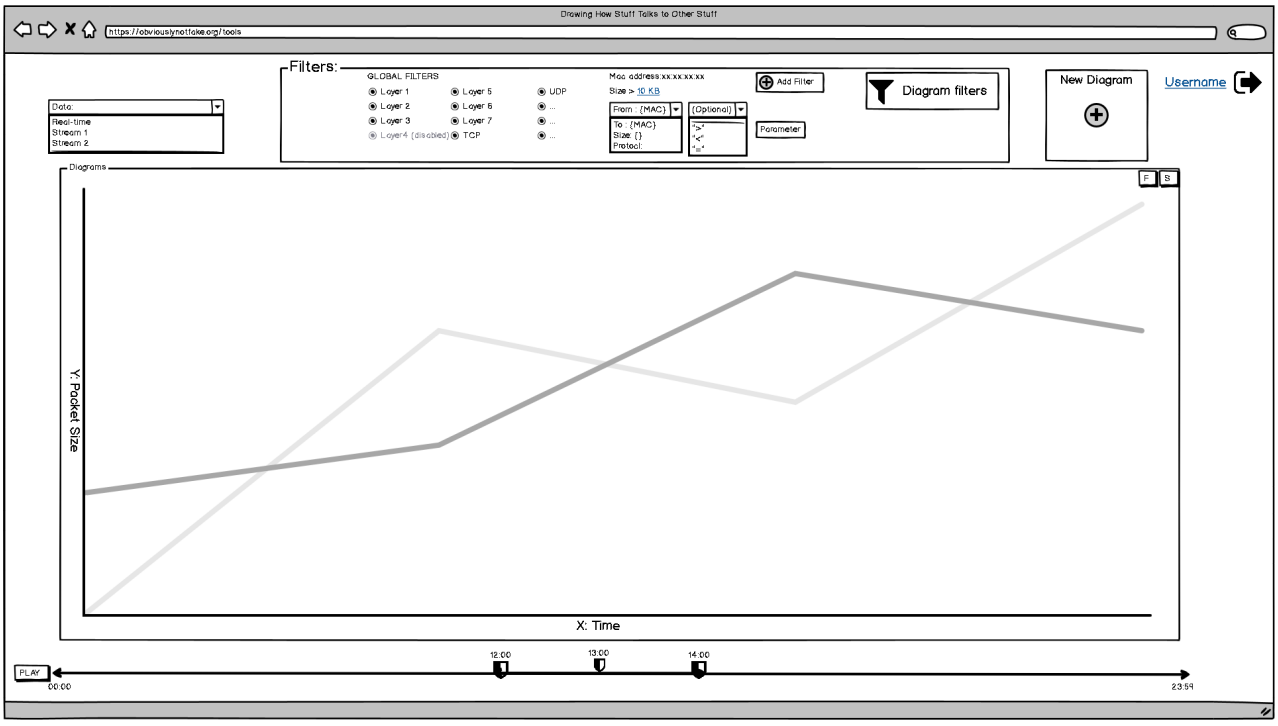


Figure 2.2: The Main window the user sees once he logs in

To discuss in depth this interface it has been divided into subsections labeled in Figure 2.3, these are:

1. The Data stream drop down menu allows the user to select which stream he is currently listening to. This dropdown menu shows the selected diagram's selected data stream and by default is set to real-time.
2. Here the user can set filters affecting all diagrams, namely limiting which layers and protocols are currently being shown. To add a new filter a user would click on the add filter button, select which part of the data stream is to be used in the filter, any optional comparators, and set a parameter in the text-box.
3. This button opens the diagram filter list. This list contains filters specific to each diagram.
4. By clicking this button the user can create a new diagram. The creation workflow is shown in Figure 2.4 to Figure 2.3.
5. Here is the currently logged in user, by clicking the button to the right of his name the user can log out.
6. This is the diagram container, inside are all diagrams the user has created with all the set constraints. At most 4 diagrams are shown in the container, and more can be made visible via the slider on the right of the container.
7. This is a diagram.
8. By hovering with the mouse over a data point a tooltip is shown with all data associated with this data point.
9. These buttons control whether a diagram is fullscreen and the diagram settings. By clicking the button labeled F the diagram grows to take on all available space in the container, like shown in Figure 2.2.
   Clicking on the button labeled "S" replaces the diagram with the Settings for this diagram like shown in Figure 2.4
10. The play button starts and stops auto scroll along the selected time frame
11. This is a timeline represented as a slider. It shows on the left and right bottom labels the beginning and end of all data streams.
    The labels on top show the currently selected time window and are movable to increasea nd decrease the time window.
    The middle shield icon shows the current time while being played, and is also movable to scroll by data manually.

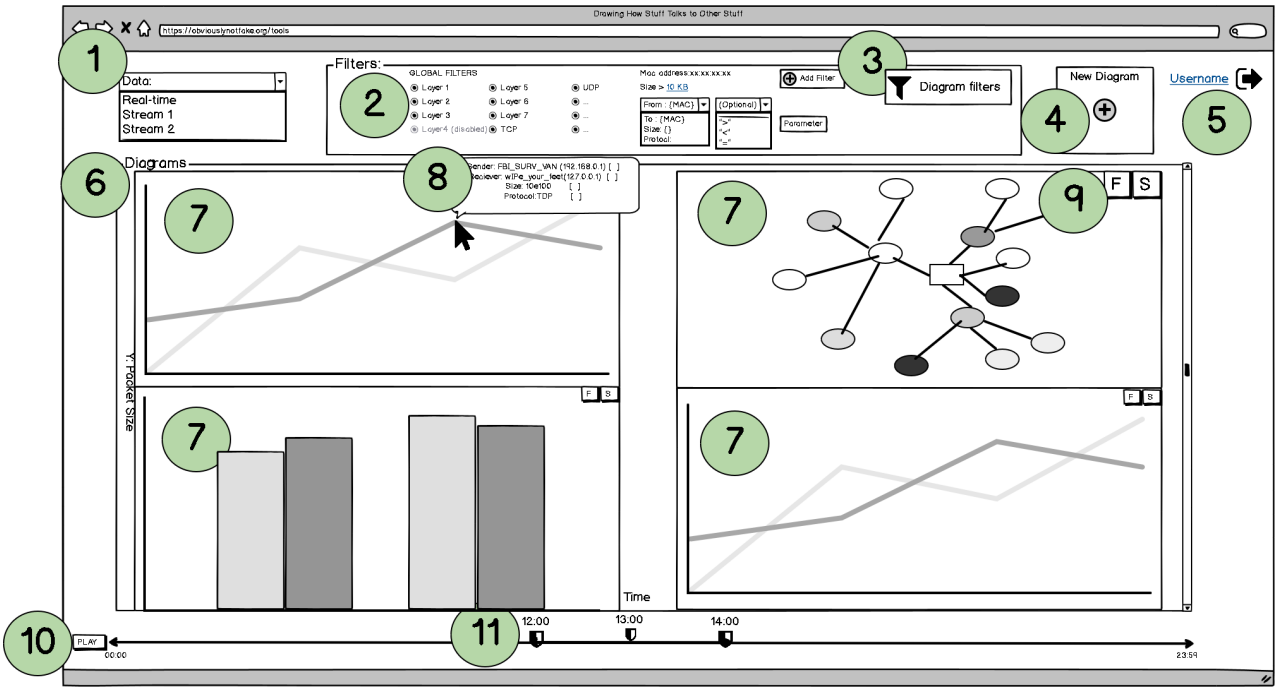


Figure 2.3: The GUI divided into relevant sections

Next we'll take a look at the workflow a user will go through when opening a new diagram.
As referenced above Figure 2.2 is what the user first sees. When he wants to create a new diagram, by clicking the button labeled new diagram, the Main window is split (shown in Figure 2.4) and the new diagram settings are shwon where the new diagram will be.
The user sets here the settings for the new diagram, these are:

- Which data stream it draws its data from
- Which data is pulled to be represented as the x and y axis
- Any filters the user wants to apply at the start

Afterwards, the screen looks like Figure 2.5, the diagram container split in half. By going through this process two more times the diagram container fills up, containing four diagrams total (shown in Figure 2.6); at this point the diagrams have their minimum size.
Any diagrams created afterwards are spawned underneath the existing ones and the user can scroll up and down to view all of them (shown in Figure 2.7)

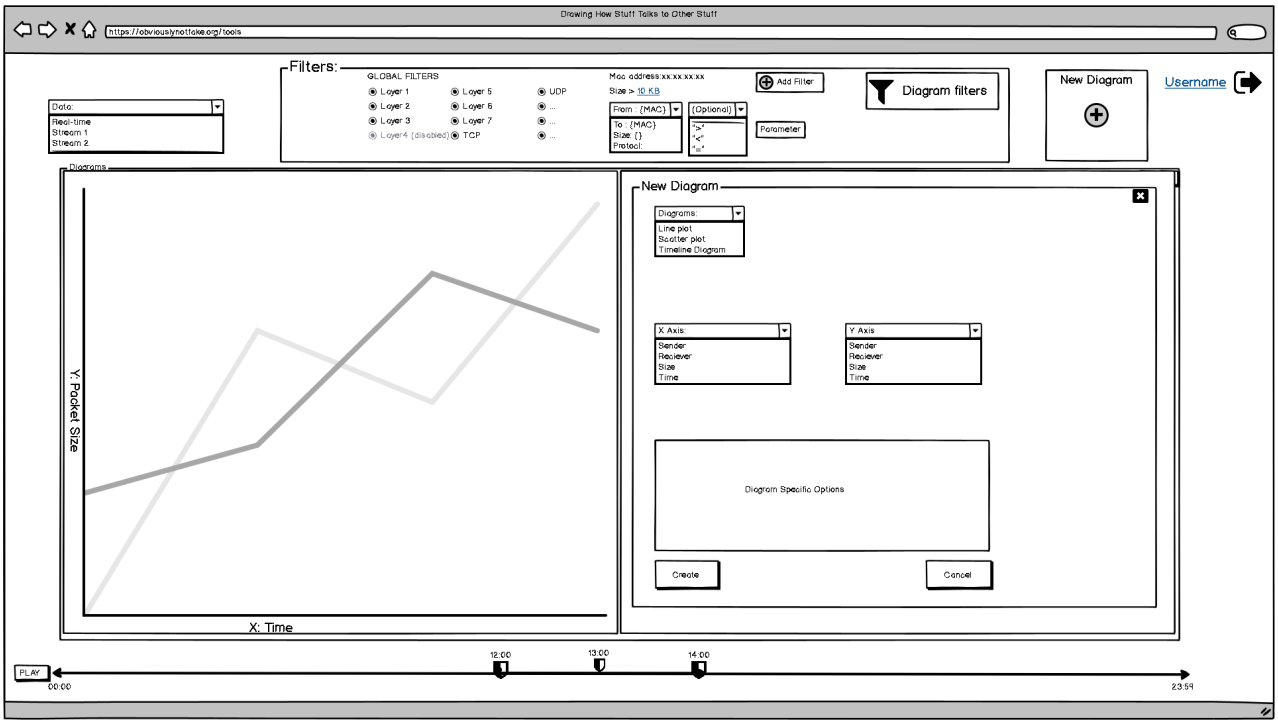


Figure 2.4: Main window split in two after user has clicked on the "New diagram" button. New diagram options are set up on the right.

Another activity the user will use frequently is the filtering system.
In the previous section we took a look at the Main window and its different sections. There we described that filters that apply to every diagram are shown at all times shown as item 2. But what about when the user wants to create a filter that only affects one diagram?
For that, the user would click on the Diagram filter (see previous section), an overlay would cover

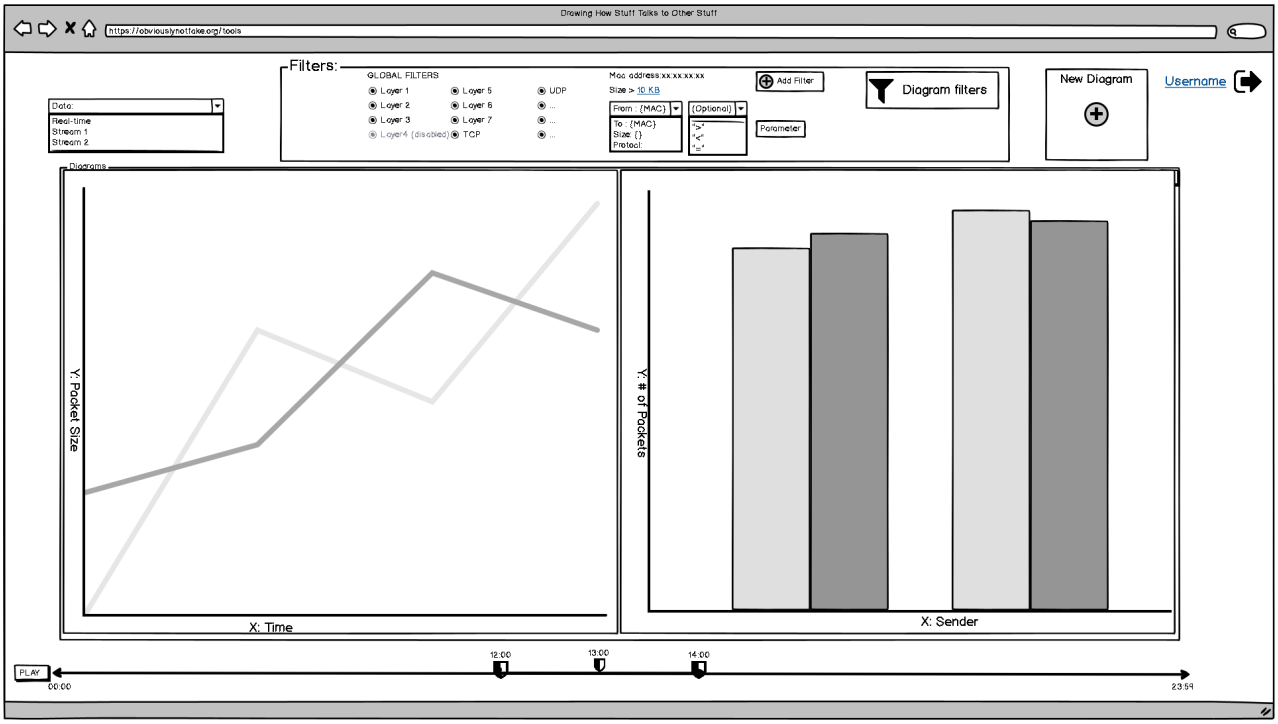


Figure 2.5: Main Window with two diagrams side by side.

the whole window and a new set of containers, similar to the global filter, would appear, as seen on Figure 2.8
On the top we see the global filter list, below it we see the filter list corresponding to the first diagram (diagram numbering is done left to right and top to bottom).
In order to set a new filter that applies to only one diagram the user would open this screen, search for the corresponding diagram number and click on the "add filter button" at which point the button get's moved down to make space for a drop down menu containing all types of data from a data stream. The user selects one, and depending on which one, if appropriate, another drop down menu appears, with contextual relational symbols (i.e. less than (<), equals, etc.) and next to it a text box where the user can input the matching string to which to filter.

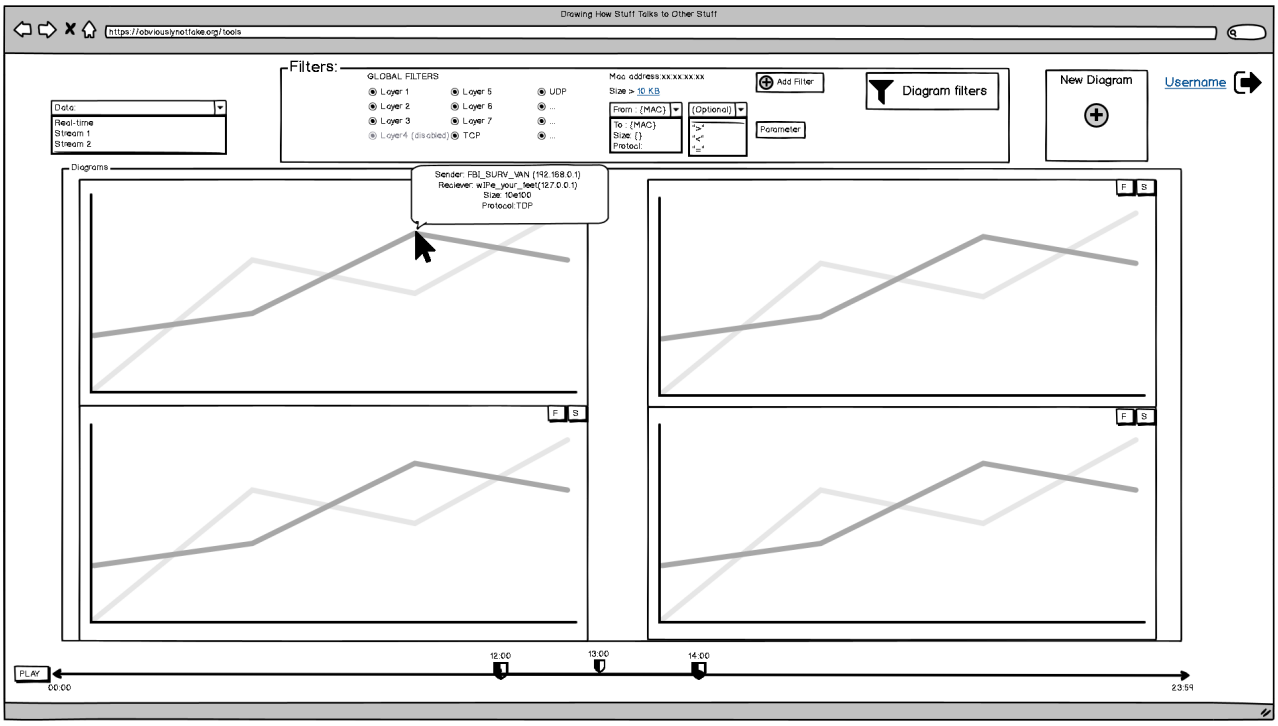


Figure 2.6: Main window with four diagrams open

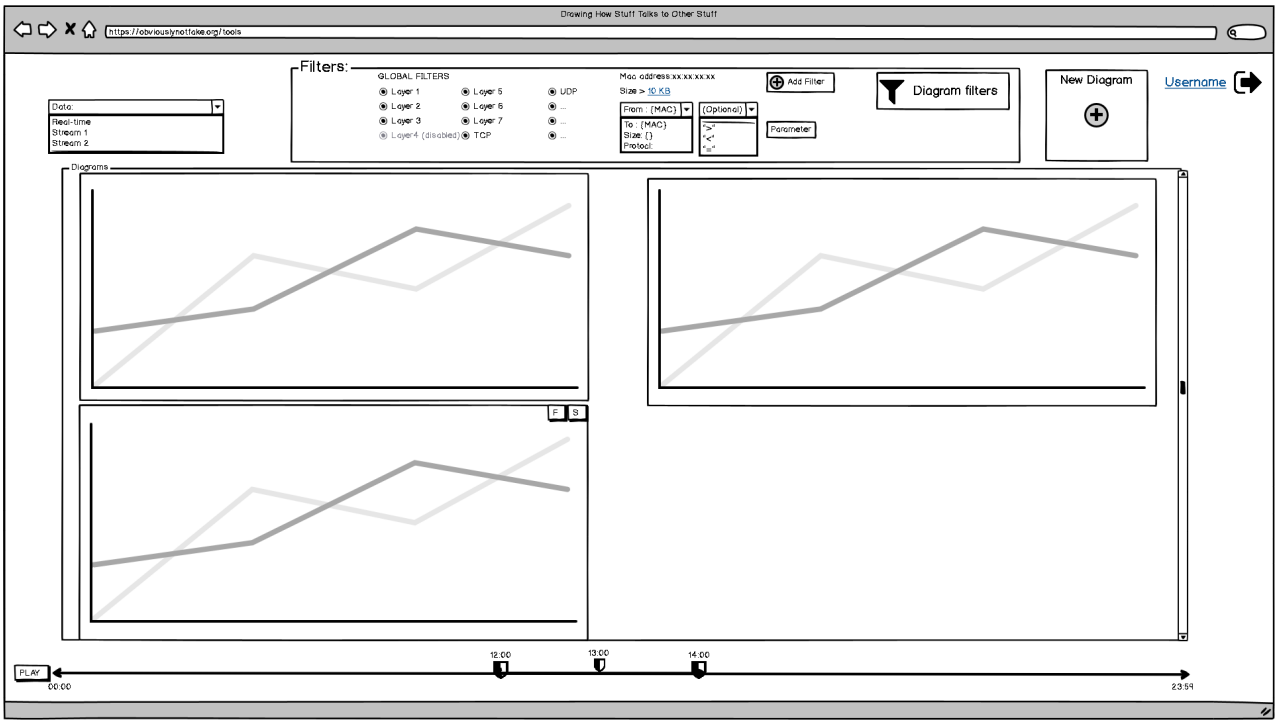


Figure 2.7: Main window with more than four diagrams open, more diagrams can be shown using the scroll bar

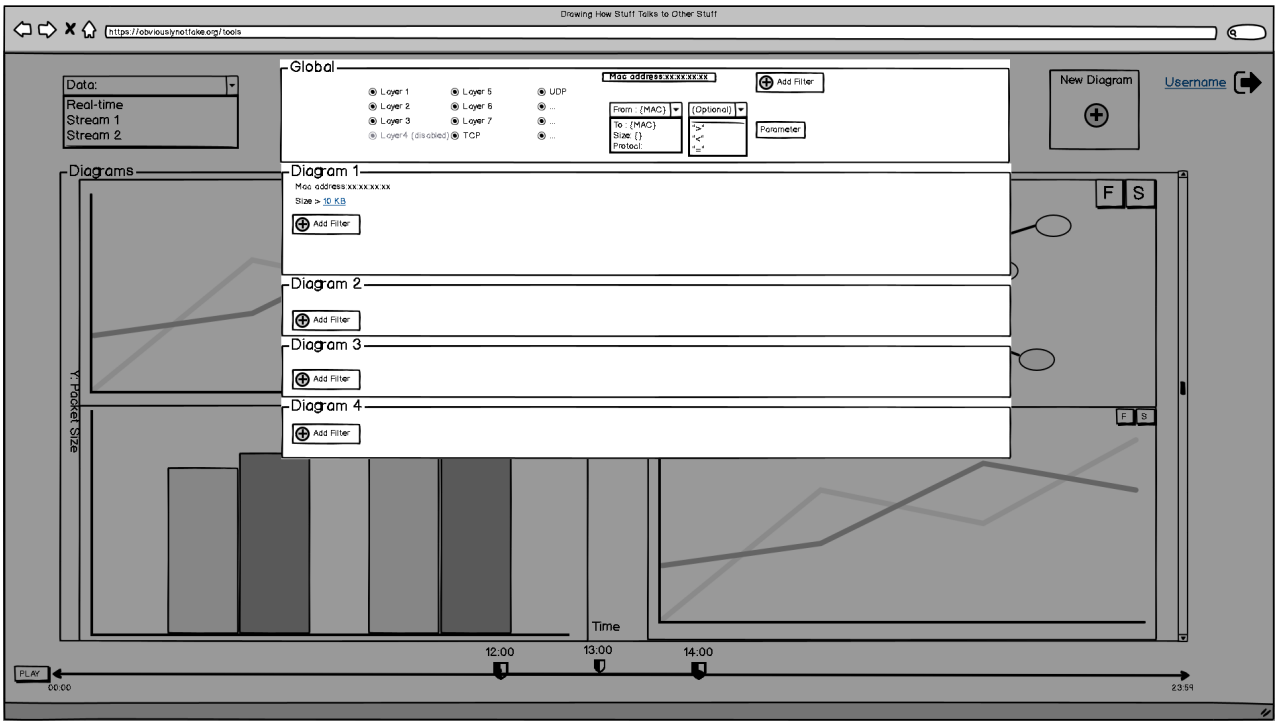


Figure 2.8: Main window with more than four diagrams open, more diagrams can be shown using the scroll bar

2.8.1.2 Diagram types

Next we'll discuss the four basic diagram types that comprise the minimum scope of this project and their possible use cases. These are Figure 2.9:

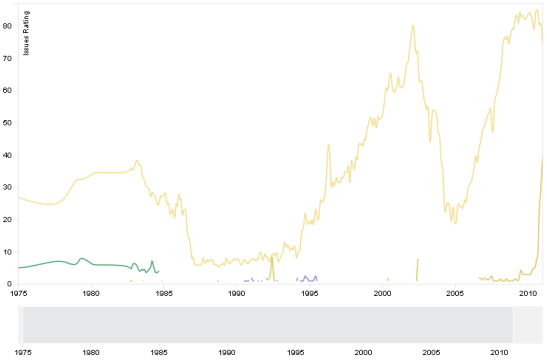
Line chart: to easily compare, linearly, two variables from a data stream. Figure 2.9(a)

Network Diagram: provides a simple way to visualize the network topology. This diagram also gives the user the ability to inspect the nodes of the network and select data from a data point to be used as a filter in other diagrams (refer to last section for usage example). Figure 2.9(b)

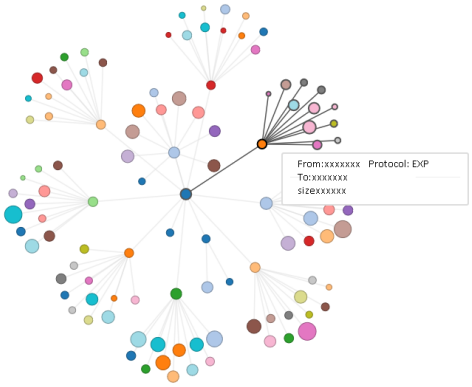
Raster spike diagram: this is useful to show the transmission window of each packet and it's corresponding partner. Figure 2.9(c)

Scatter plot: a way to visualize up to 4 dimensions of the data stream. Figure 2.9(d)

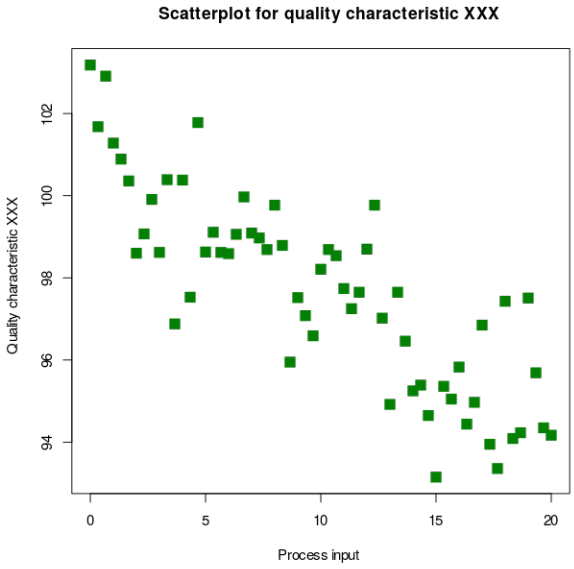
These are examples and since the user has the ability to chose what part of the data stream is used for each axis it does not convey the full ability of these diagrams, that is left to the end user.



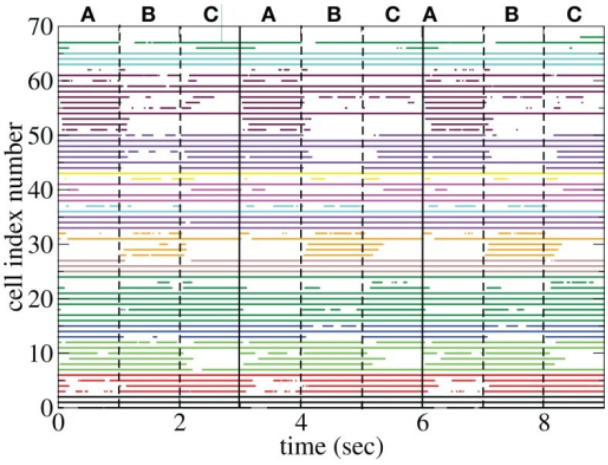
(a) Line Chart
[**Firstarticle**]



(b) Net diagram
[**Secondarticle**]



(c) Scatterdiagram
[**Thirdarticle**]



(d) Spike raster plot
[**Fourtharticle**]

Figure 2.9: Diagram types

2.8.2 Scenarios

S100: An operator wants to check manually/visually whether network nodes appeared or disappeared over the last day

- the operator opens the web page
- the operator selects the database as data source
- the operator selects a time-line-based diagram type
- the operator selects node addresses as the data to be displayed
- the operator moves to or selects the last 24 hours as the range of data to display
- the operator closes the web page

S200: A security analyst wants to look at the current flow rates between network nodes to see whether they change / there are trends

- the analyst opens the web page
- the analyst selects a source of live data
- the analyst selects an appropriate visualization type
- the analyst selects node addresses as the independent variable
- the analyst selects flow rates as the data to be displayed

S300: A security analyst wants to examine a specific point of data
Precondition: the analyst has already selected the relevant dataset and visualization type

- the analyst selects a data point
- the GUI displays a small pop-up window with all the data of this data point
- the analyst right-clicks one of the attibutes in the pop-up window and selects "Display all matching types"
- the GUI marks all data points that have the same value in this attribute

S400: The user wants to look at alarms/notifications (TBD)

- the user opens the web page
- the user selects the database as data source
- the user selects the data stream from the relevant dissector
- the GUI diplays the notifications along a timeline, according oder of occurance
- the user right-clicks on the x-axis and selects "use record number"
- the GUI diplays the notifications along a timeline adjacently

S500: The user wants to look at normal data together with alarms/notifications
Precondition: Scenario S100 apart from closing the web page

- the user selects menu "data", entry "sources"
- the GUI displays a list of all known data sources with a checkbox in front of each

- the user selects the checkboxes for the data sources they want to examine
- the GUI displays data from all these data sources within the currently active visualization

2.8.3 Use cases

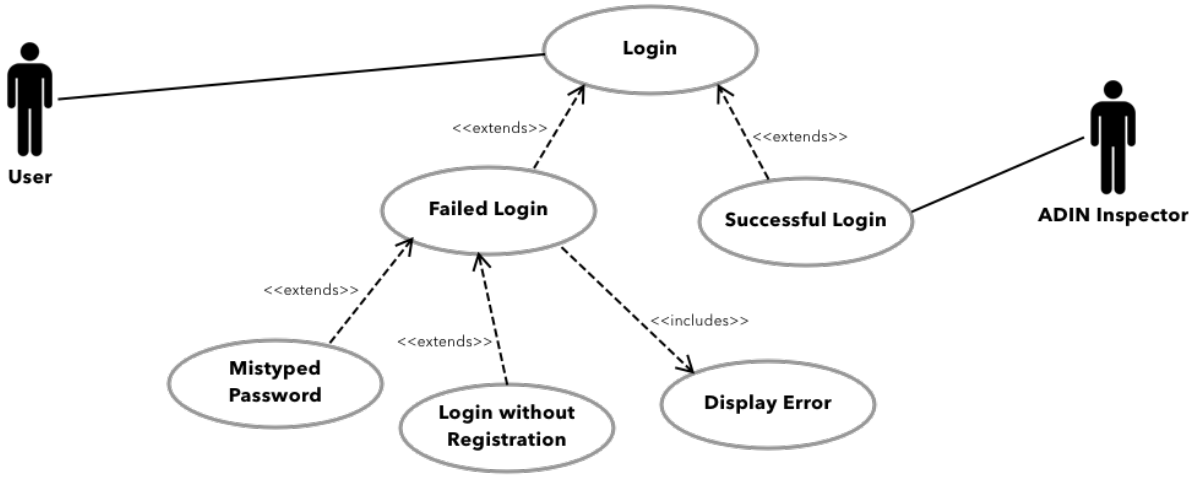


Figure 2.10: User Login.

Use Case UC1: Login

Scope: ADIN Inspector and hub

Level: user goal

Stakeholders and Interests:

- User: Wants to login quickly and easily
- System administrator: Wants to ensure that only authorized persons access the system. Wants few user support requests.
- Manager: Wants protection of data. Wants no obstacle to the user's work.

Preconditions: An account for the user has been created.

Postconditions: User is loged in.

Main Success Scenario:

1. Customer opens the ADIN website in the browser
2. Customer enters his/her username and password

3. The ADIN system checks the entered password and loads the access permissions for the user.
4. The sytem presents the ADIN main screen.
5. User begins to use the ADIN Inspector.

Exception control flow

Possible at any time: the network connection to the server fails or the server crashes:

1. the user reloads the web site which takes him/her to the ADIN login screen

At step 3: Wrong username or password:

1. System produces an error message and stays on the login screen.

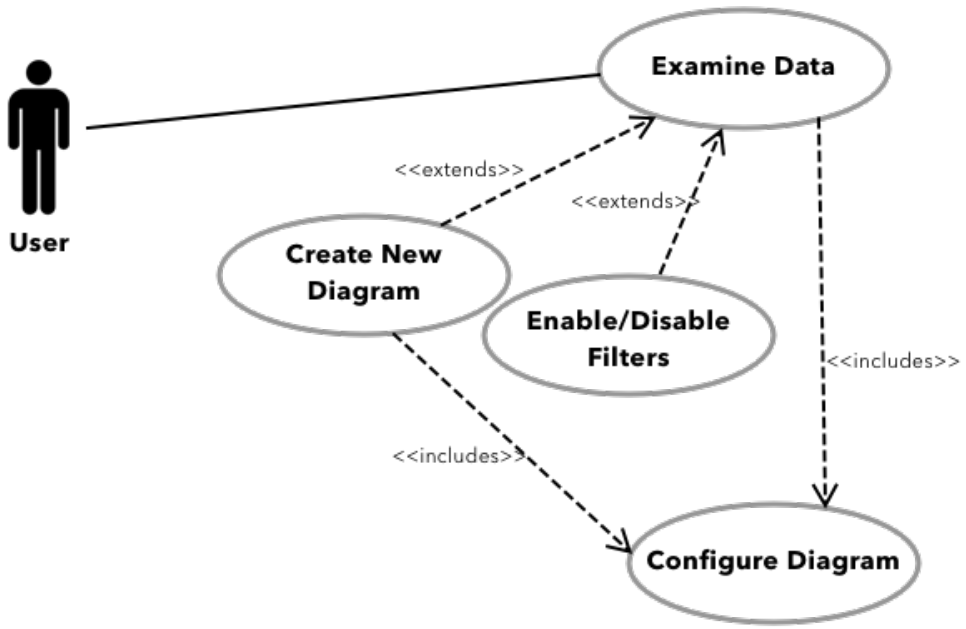


Figure 2.11: Examine Data.

2.8.3.1 Interactivity

Visual analytics methods combine interactive visualisations with automated analysis techniques. This allows the user to decide e.g. which part of the data he or she wants to explore in more detail.

A basic principle for visual data exploration was introduced by Shneiderman (1997) by what he called the “The Visual Information Seeking Mantra:

Overview first, zoom and filter, then details-on-demand". This lets the data analyst define to a certain level what he or she wants to analyze and visualise.

Similar to this, Bertin (1983) specified three "levels of reading,": The elementary level (allowing the analyst to look at the information about a single data record), the intermediate level (showing summarised information about a group of data records), and the global level (providing an overview of all data elements).

2.8.4 Object Modelling

The diagram in Figure 2.12 shows a high-level view of the components of the ADIN-Inspector. The PSE group will develop the visualization component that runs in the web browser and a server side component called hub. The hub interfaces with the data sources and also implements the access control.

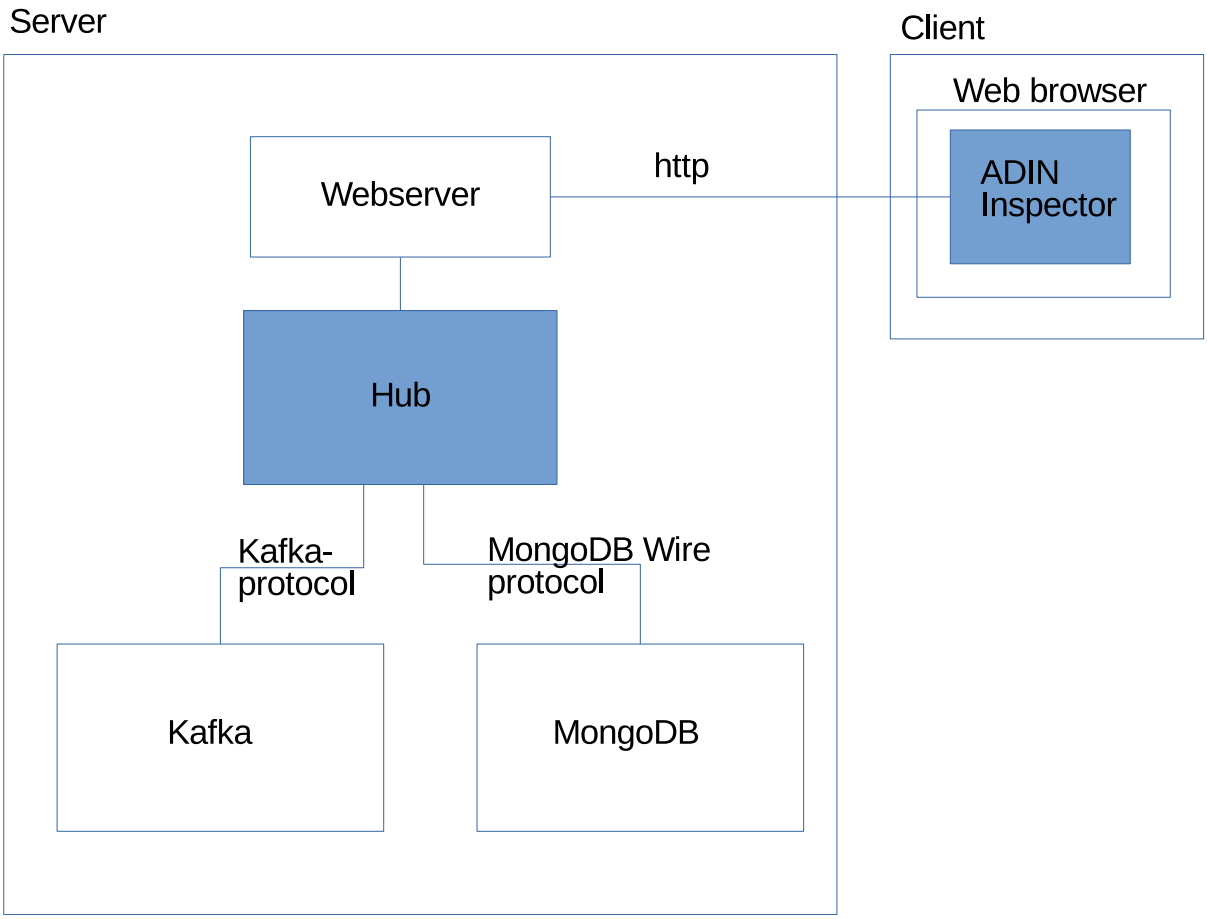


Figure 2.12: General system layout. Components to be developed by the PSE group are marked blue.

2.8.5 Dynamic Modelling

The sequence diagram in Figure 2.13 shows the actions of the Hub component during login.

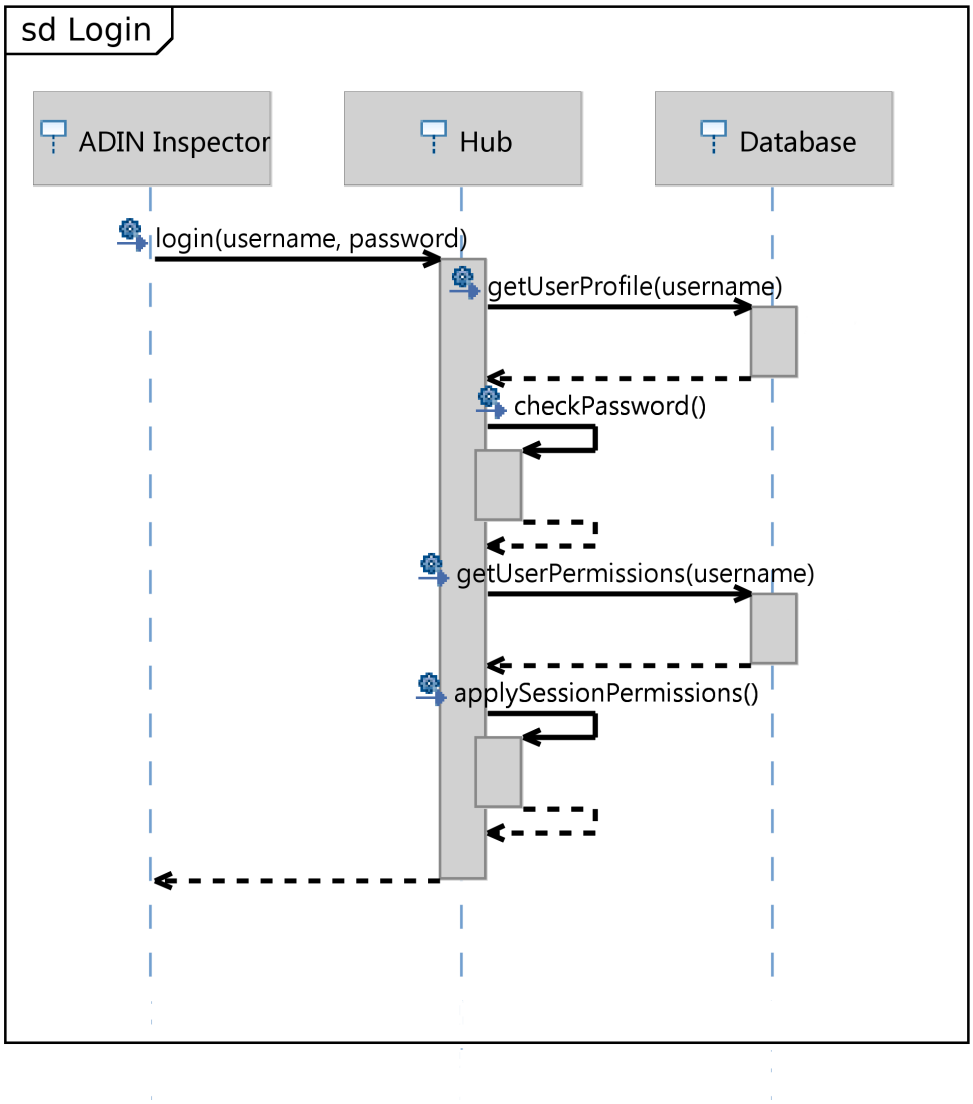


Figure 2.13: The activity of the Hub during login.

2.9 Glossary

3 Design

3.1 Front-End

This subsection describes the front-end of the ADIN INSPECTOR - the UI elements the GUI consists of, and how states are handled. A series of final UI design mockups are presented under UI Design Mockups subsection, whereas an overview of the GUI classes can be seen in Figure 3.8.

The GUI elements are implemented in React and Material UI, whereas the internal logic and application state management are written with MobX.

3.1.1 UI Design Mockups

An early stage interactive demo is available at `https://adin-frontend.netlify.com`.

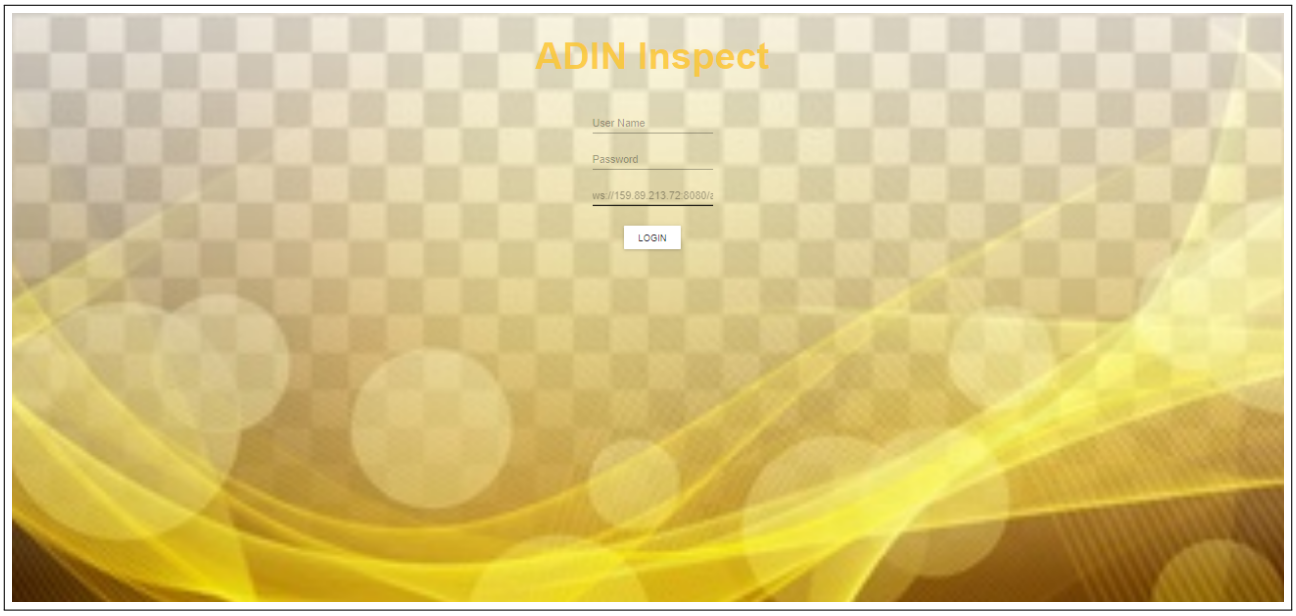


Figure 3.1: Login screen

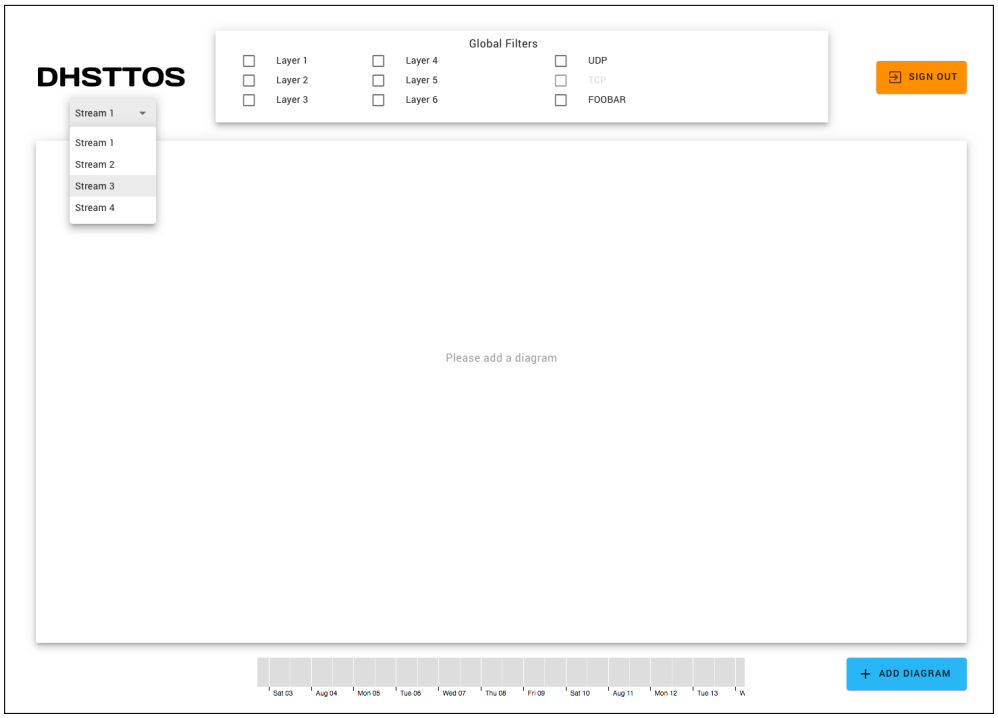


Figure 3.2: Initial empty screen

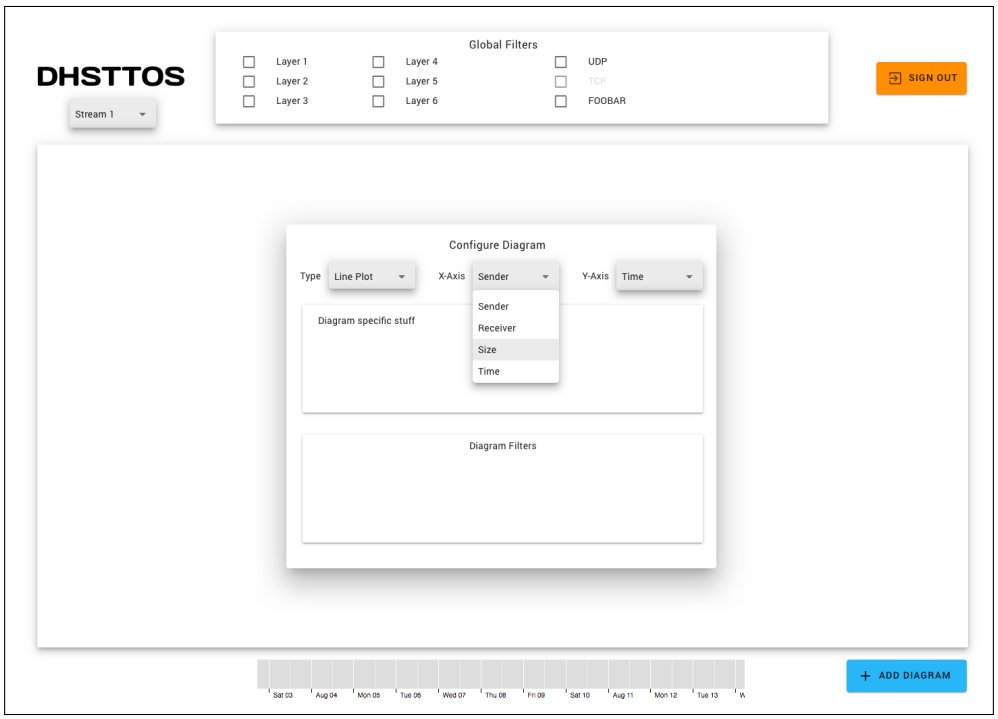


Figure 3.3: Adding first diagram

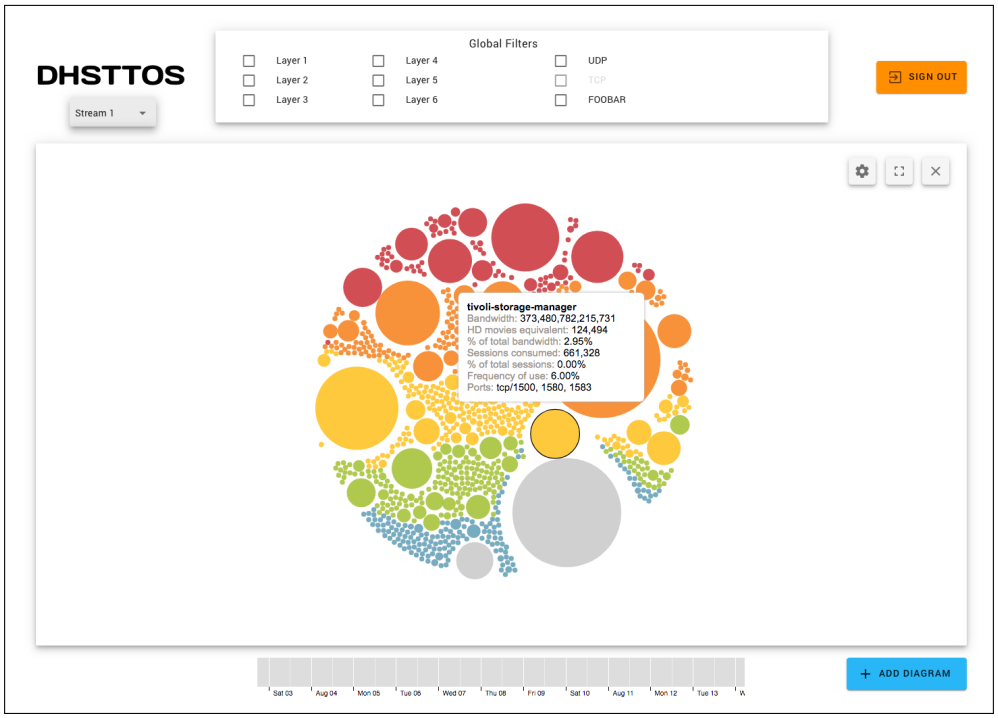


Figure 3.4: Displaying a single diagram

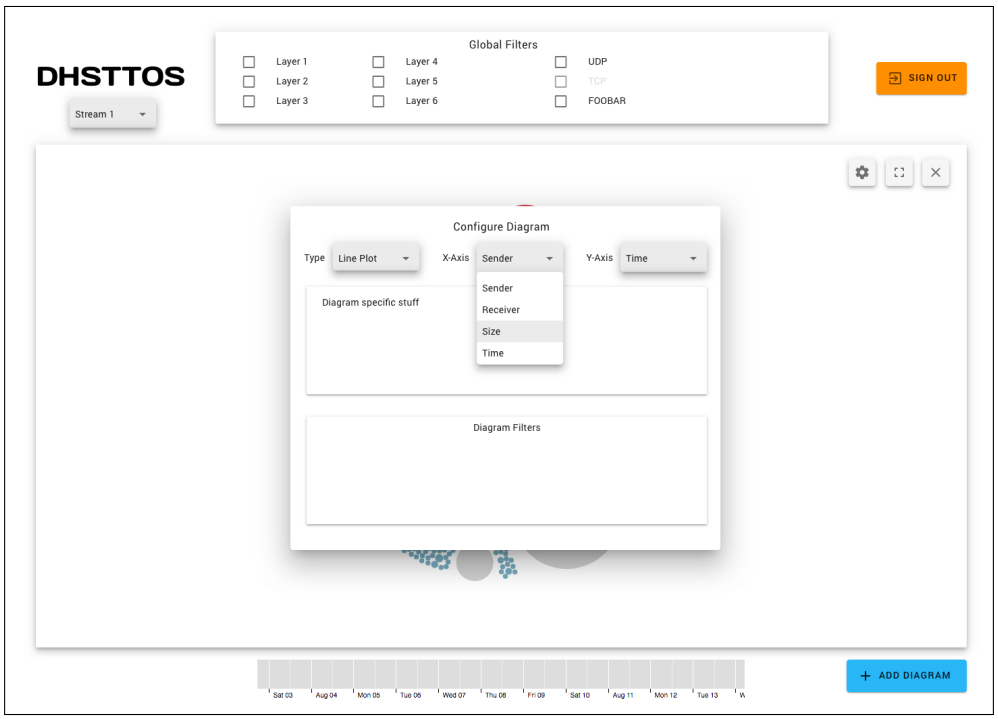


Figure 3.5: Adding new or configuring existing diagram

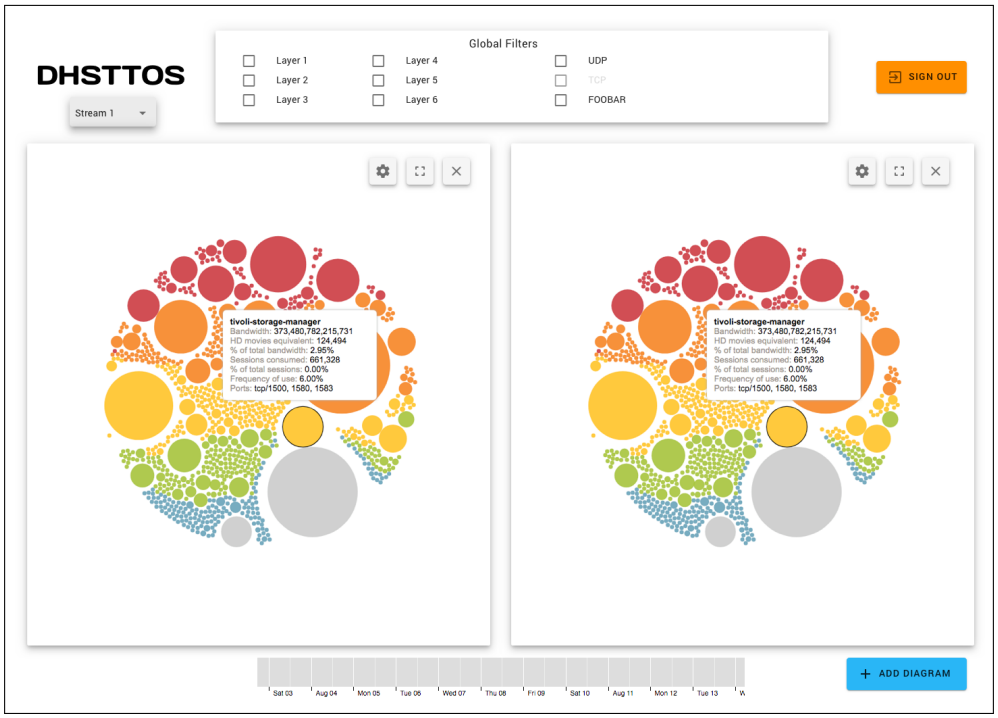


Figure 3.6: Displaying two diagrams

Figure 3.7: Adding additional or configuring existing diagram

3.1.2 Class Diagrams

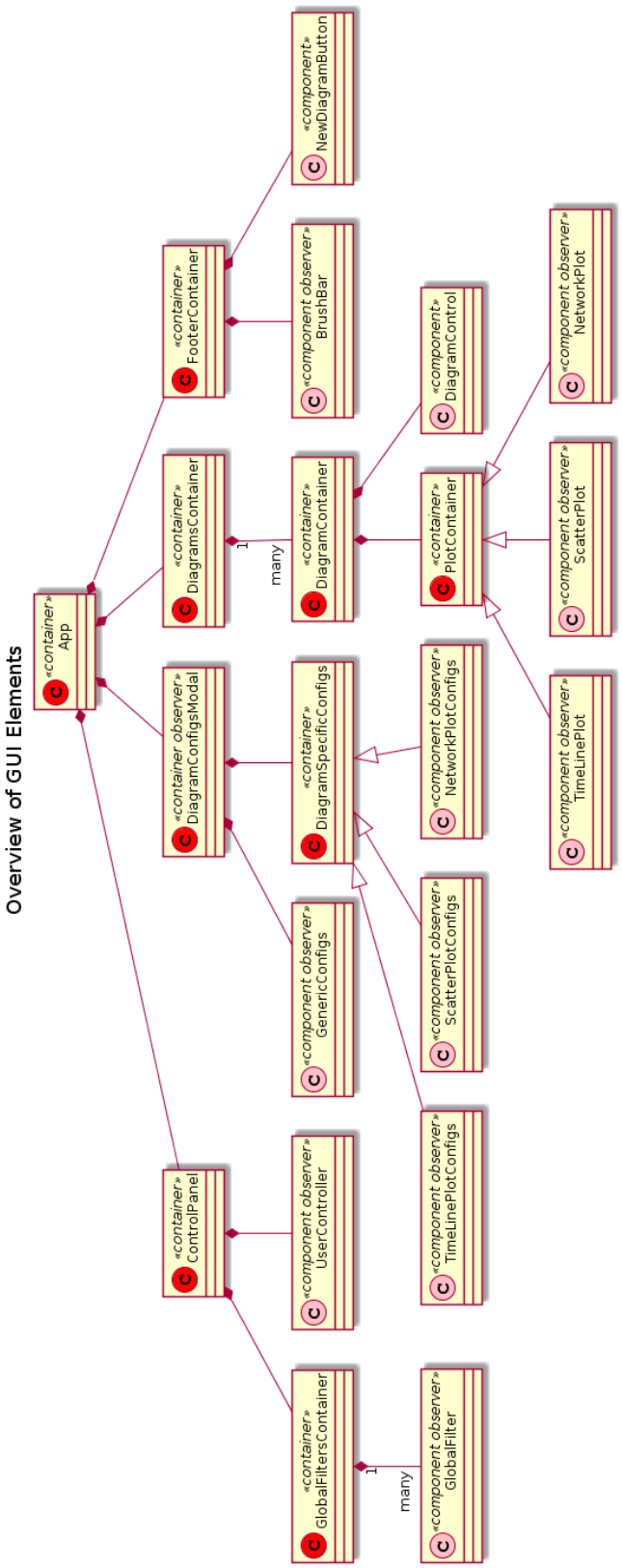


Figure 3.8: This diagram shows an overview of GUI elements and their relationships inside the main application, when the user has successfully logged in.

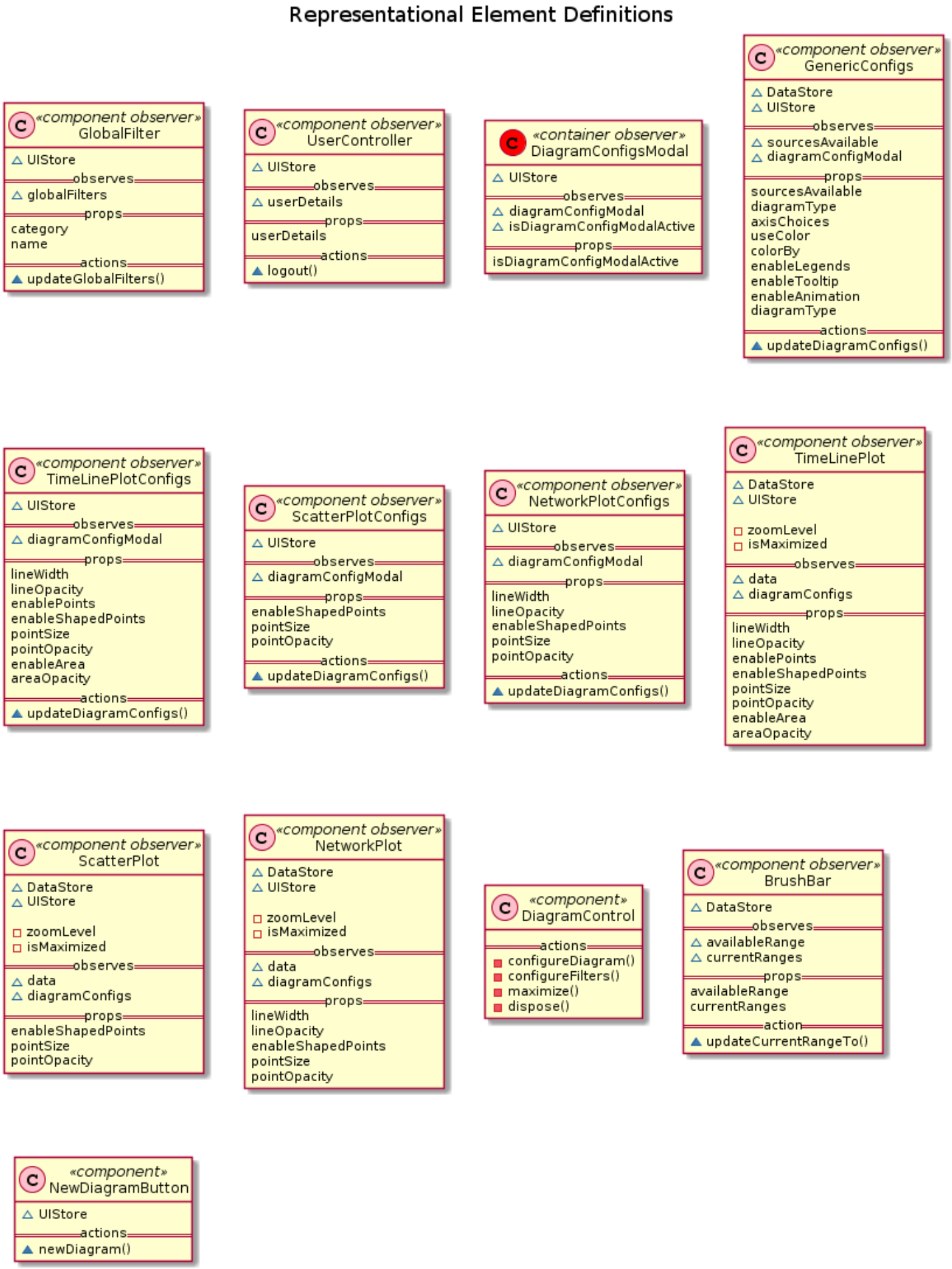


Figure 3.9: This diagram shows the definitions of all representational elements.

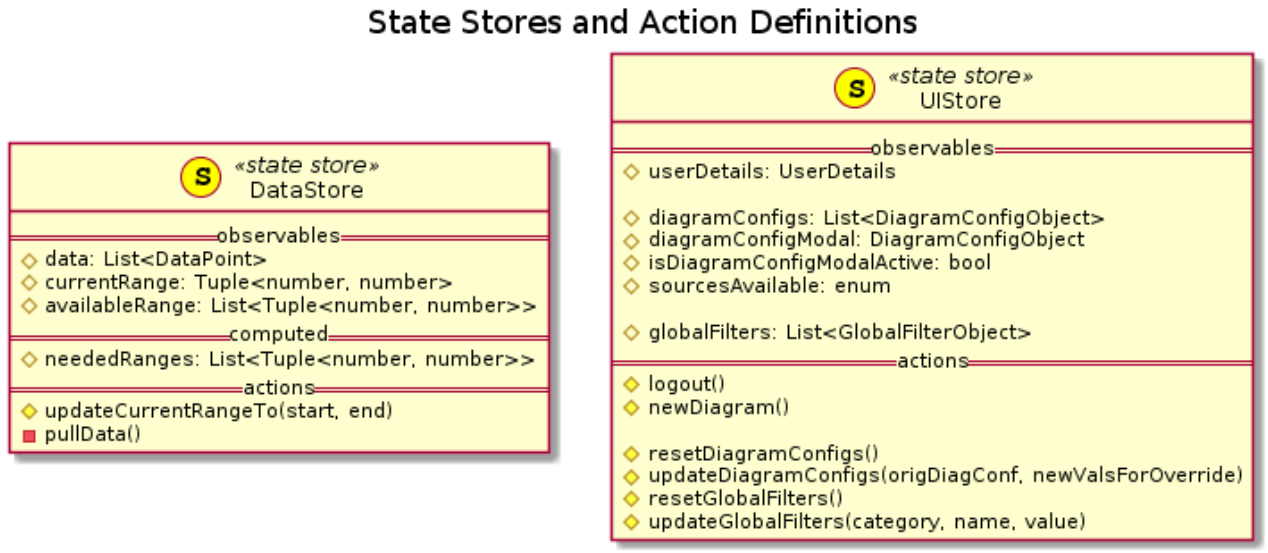


Figure 3.10: This diagram shows the design of the MobX state store objects and predefined actions to mutate the states.

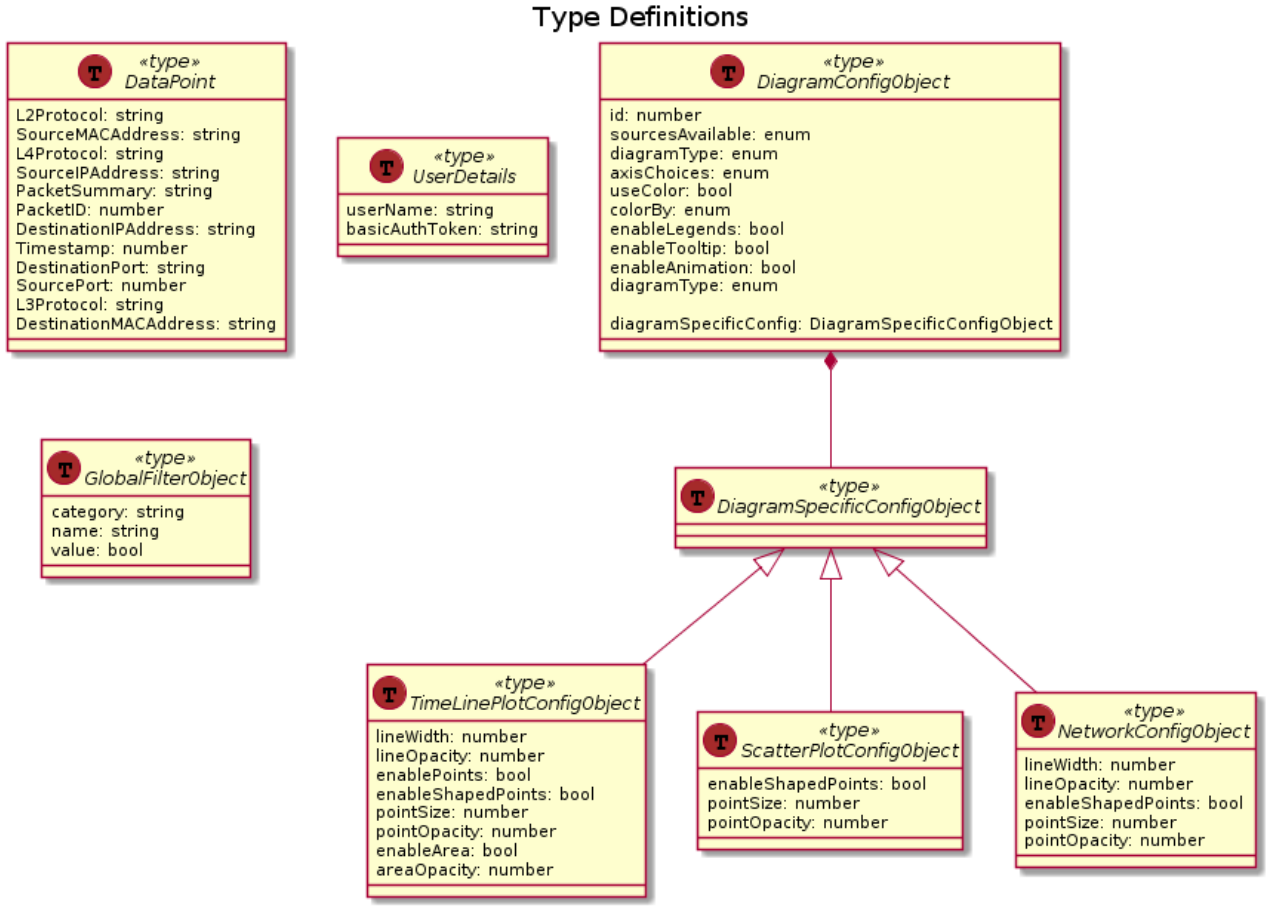


Figure 3.11: This diagram shows the definitions of custom types that are used in the MobX state stores.

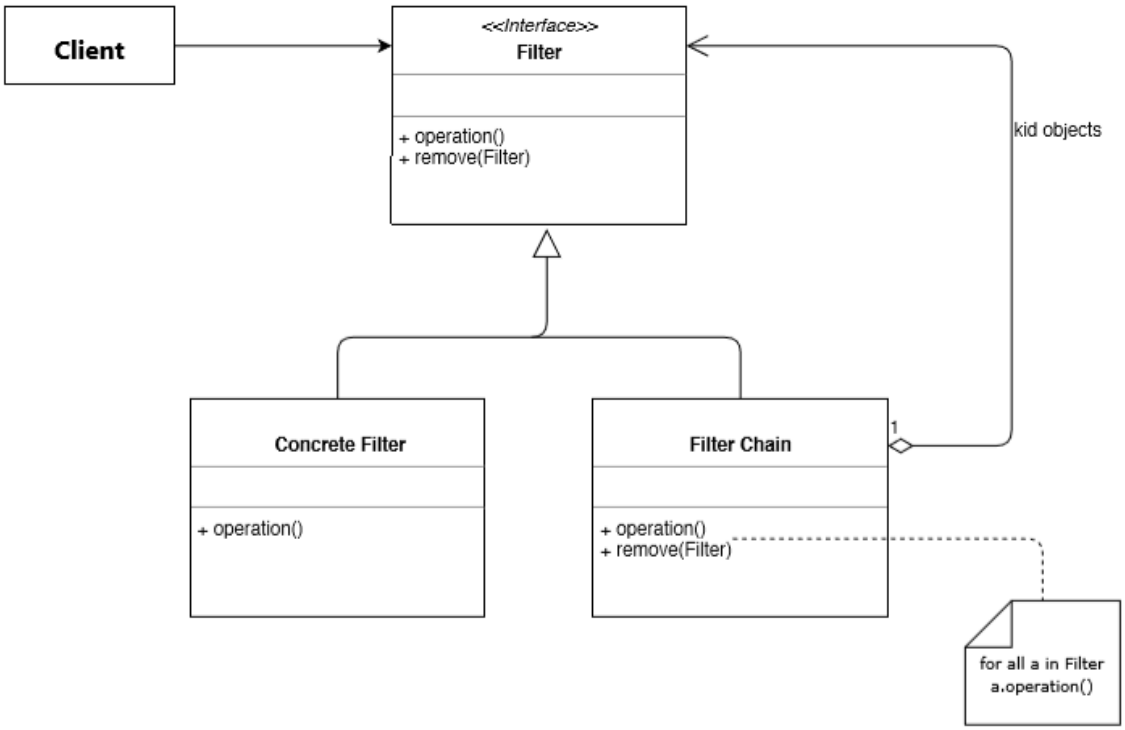


Figure 3.12: Class diagram representing the filtering chain.

3.1.3 Sequence Diagram

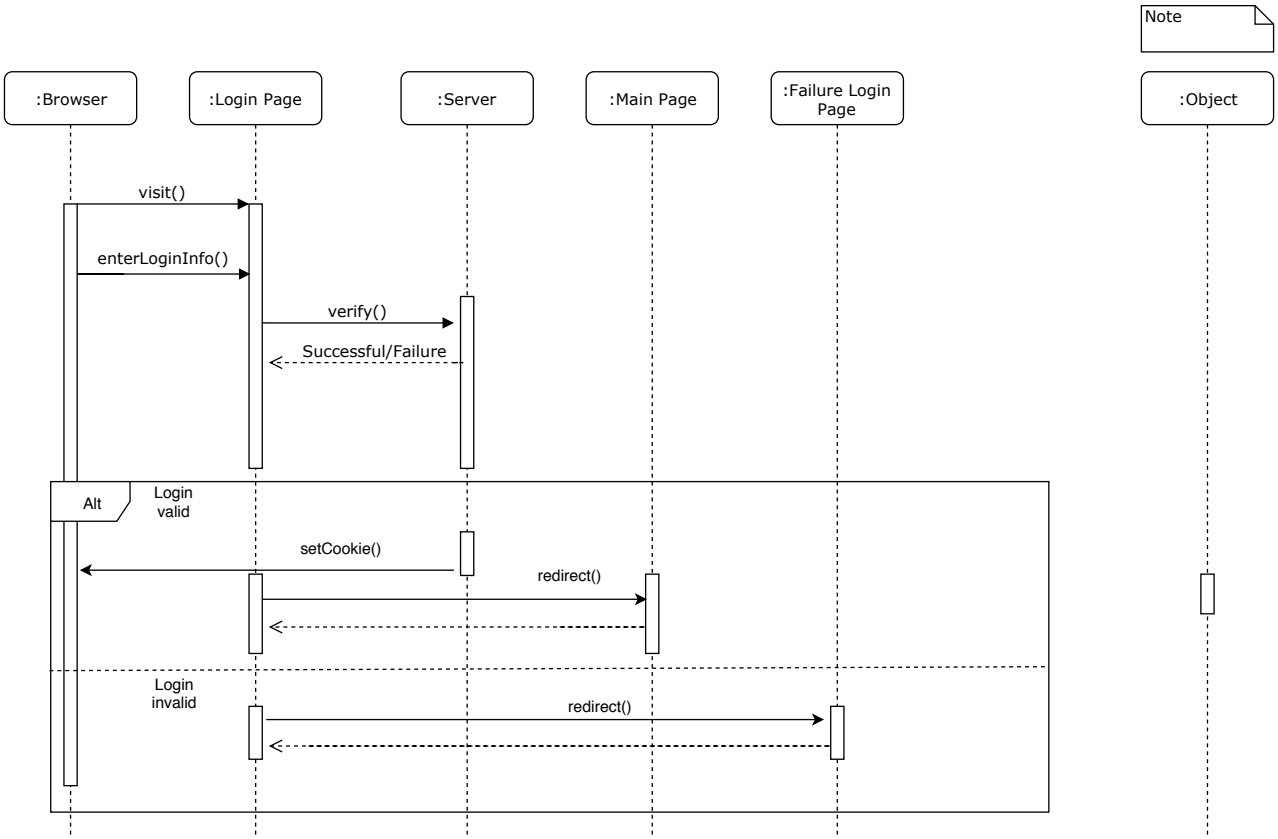


Figure 3.13: Sequence diagram of Login Authentication System. The user credentials are transmitted via https and the server returns a token to identify the following user session.

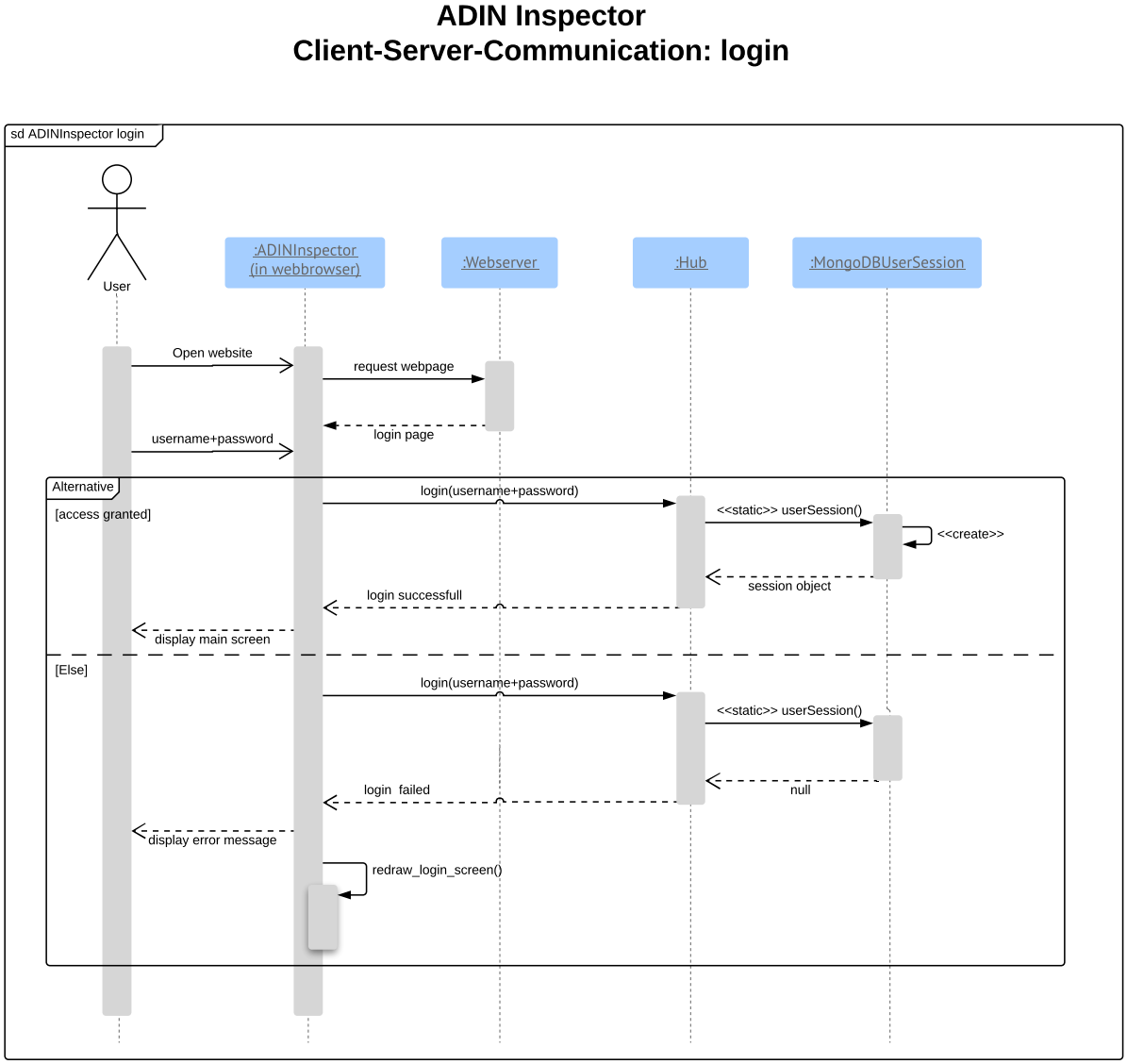


Figure 3.14: This diagram shows an alternative view of the login sequence where the user credentials are transmitted via the same encrypted WebSocket connections which is used for the following session.

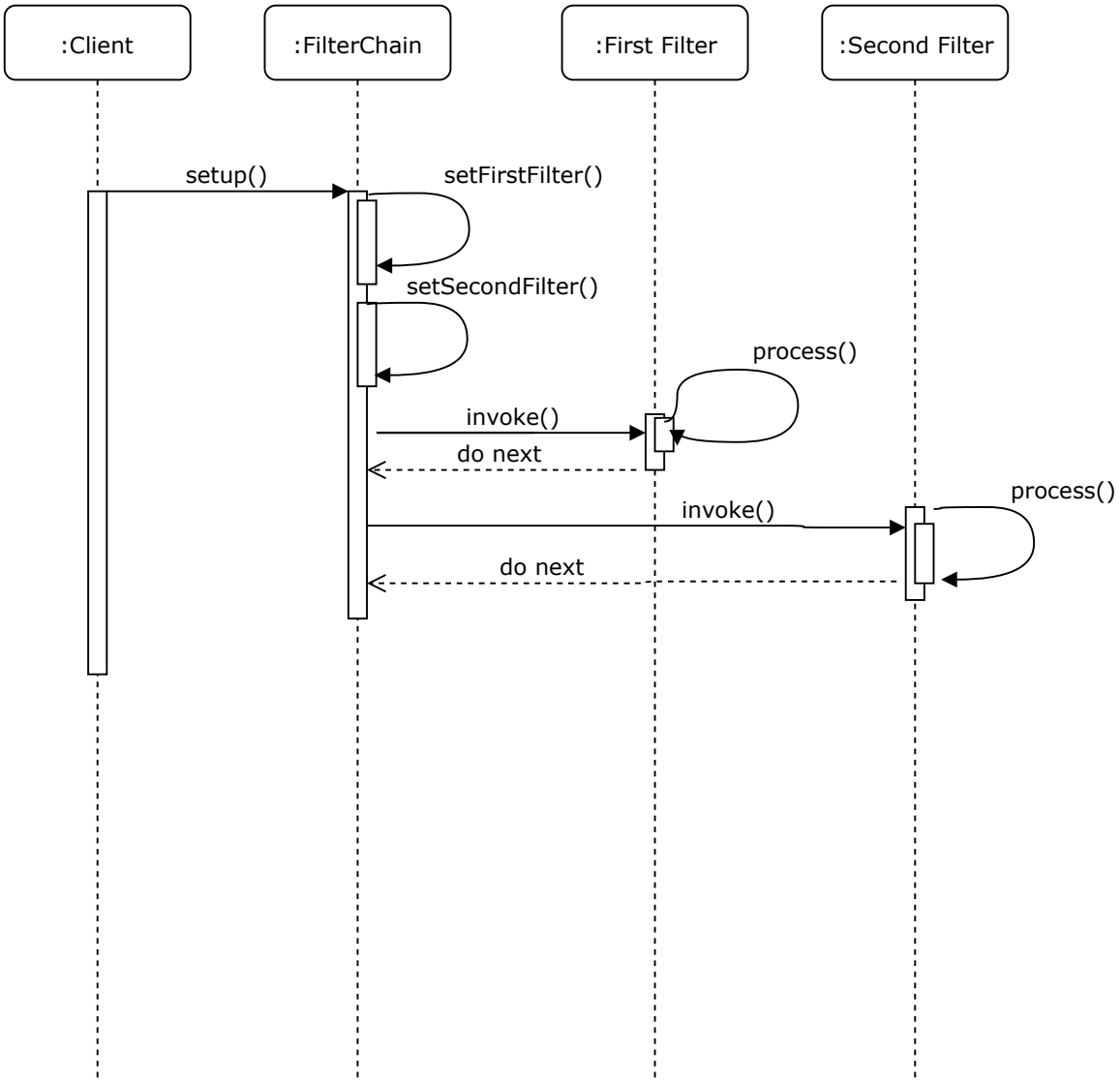


Figure 3.15: Sequence diagram of chaining filters in a specific diagram.

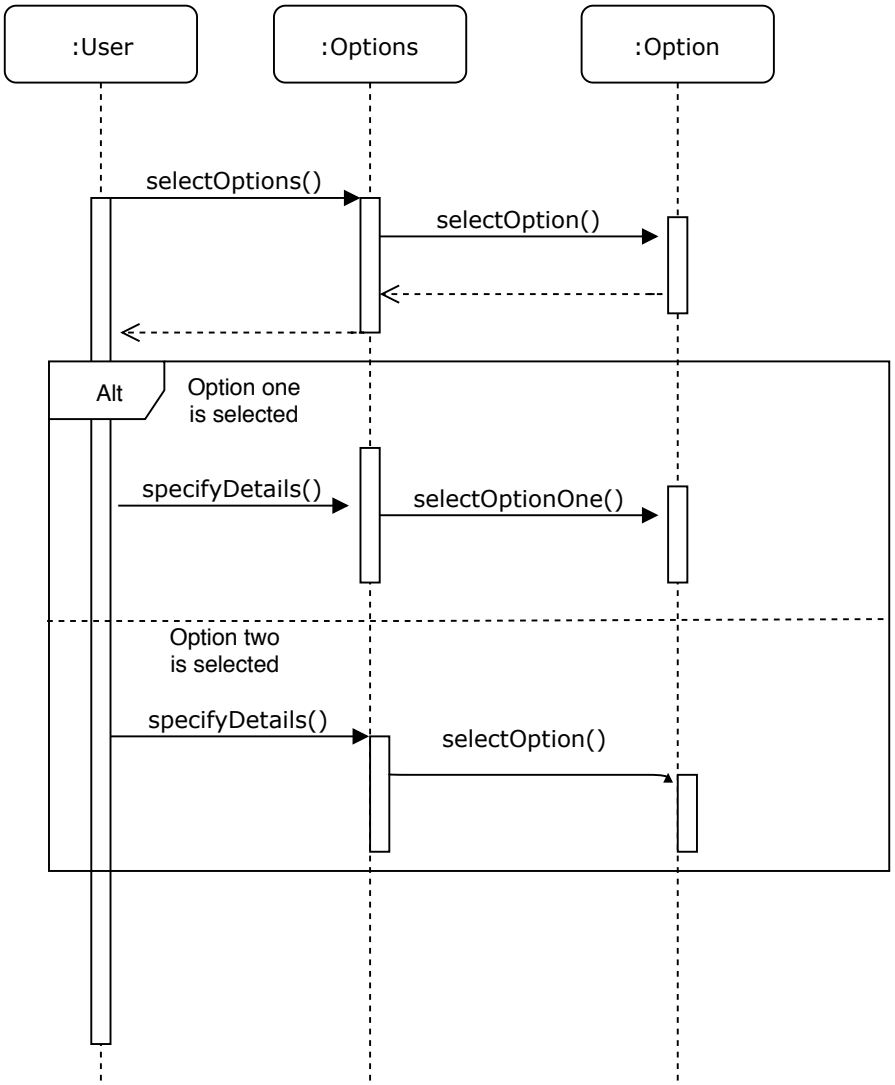


Figure 3.16: Selecting a specific option from a drop-down menu.

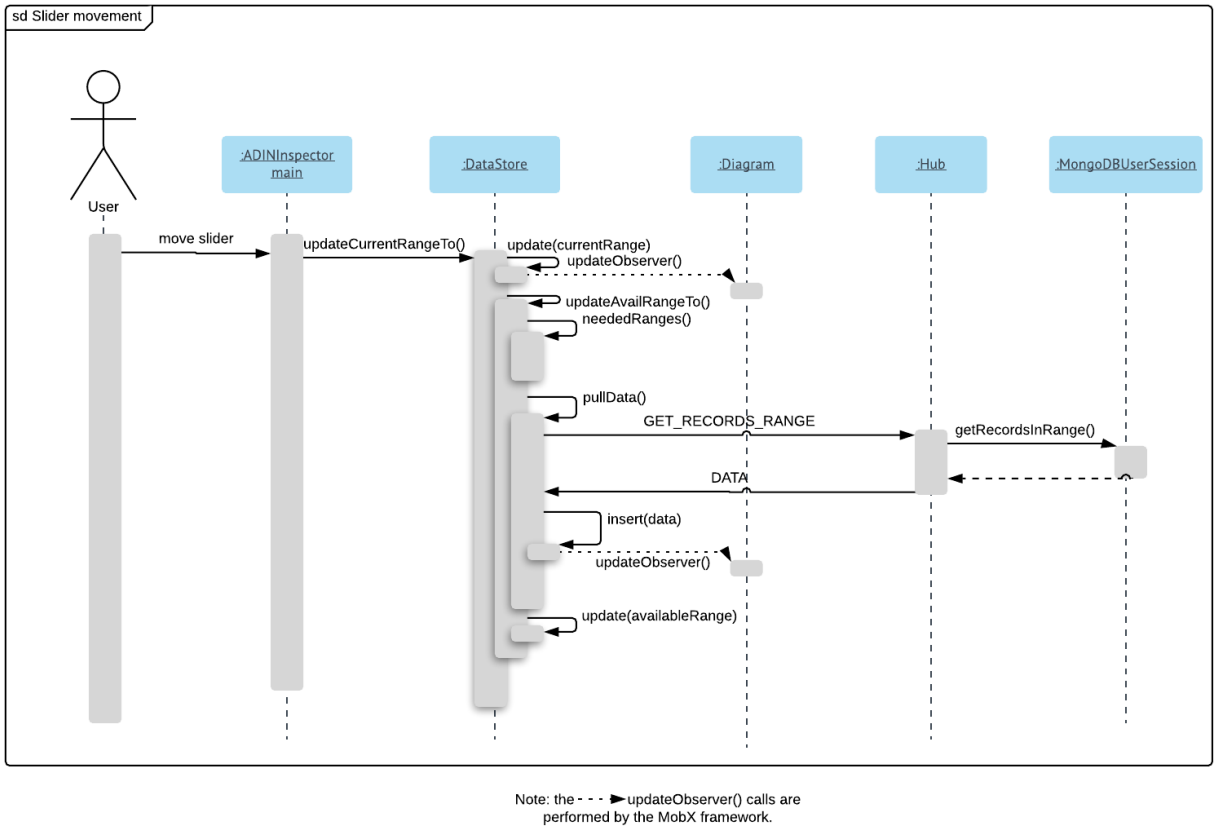


Figure 3.17: Sequence diagram showing control flow for handling a movement of the slider by the user.

3.2 Client-server protocol

The client (web browser) and server communicate using the encrypted WebSocket protocol (wss://). Messages between client and server are exchanged as strings in JSON format. Communication is typically initiated by the client. Each request has a message id which is echoed back in the response to help the sender to link the response to the request. The ids can be random, they don't have to be in a specific order.
By default, all requests apart from login will be ignored. A communication session starts with a login request that is responded to with an "OK" session control message. Within a communication session, login is ignored and all other requests can occur in arbitrary order. On receiving a logout request, the server returns into the default state, i.e logged out.
The user credentials that the user enters in the login web page can be transmitted in two different ways. In the first case they are sent via https with the POST method to the web server and the server returns a token which the client (the web browser) will save in a cookie and then transmit to the server in the WebSocket session with the LOGIN_TOKEN message to authenticate the WebSocket session (see Figure 3.13). In the alternative case the client opens an encrypted WebSocket connection to the server and sends the user credentials with the LOGIN message; i.e user authentication and data communication take place within the same WebSocket session (see Figure 3.14).
All communication, including the user authentication, uses https or wss, respectively, both of which use TLS/SSL encryption. The MongoDB data base doesn't store user passwords in plaintext but stores a hashed value derived from the password.
Responses carrying data, specifically "data set" and "collection size", will echo back any key-value pairs that are in addition to those specified in the protocol. This allows the client to add information that will help in processing the response, e.g. when requesting several "chunks" of data that will be concatenated.
In the following list words in angle brackets ("<>") are placeholders.

3.2.1 Requests from client to server:

- login
  syntax: {"cmd": "LOGIN", "user": "<username>", "pwd": "<password>", "id": "<id>"}
  expected response: session control with key "par" having the value "LOGIN"
- authentication via token
  syntax: {"cmd": "AUTH", "token": "<token>", "id": "<id>"}
  expected response: session control with "par": "AUTH"
- logout
  syntax: {"cmd": "LOGOUT", "id": "<id>"}
  expected response: session control with "par": "LOGOUT"
- getAvailableCollections
  syntax: {"cmd": "GET_AV_COLL", "id": "<id>"}
  expected response: list of collections
- getCollectionSize(collection)
  syntax: {"cmd": "GET_COLL_SIZE", "par": "<collection>", "id": "<id>"}

where <collection> is the name of a collection
expected response: collection size

- getStartRecord(collection)
  syntax: {"cmd": "GET_START", "par": "<collection>", "id": "<id>"}
  expected response: single data point, with "idx": "start" and key "data" containing the first data point of the given collection

- getEndRecord(collection)
  syntax: {"cmd": "GET_END", "par": "<collection>", "id": "<id>"}
  expected response: single data point, with "idx": "end" and key "data" containing the last data point of the given collection

- getEndpoints(collection)
  syntax: {"cmd": "GET_ENDPOINTS", "par": "<collection>", "id": "<id>"}
  expected response: data endpoints, with the key "data" containing a array consisting of the first and the last data point of the given collection

- getCollection(collection)
  syntax: {"cmd": "GET_COLL", "par": "<collection>", "id": "<id>"}
  expected response: data set

- getRecordsInRange(collection, key, start, end)
  syntax: {"cmd": "GET_RECORDS_RANGE", "par": "<collection>", "key": "<keyvalue>", "start": "<startvalue>", "end": "<endvalue>", "id": "<id>"}
  where <key> is the name of a key in the given collection and <startvalue> and <endvalue> are valid values for this key
  expected response: data set, where key "data" contains all data points (i.e. records) from the given collection for which the given key is in the interval [start, end).

- getRecordsInRangeSize(collection, key, start, end)
  syntax: {"cmd": "GET_RECORDS_RANGE_SIZE", "par": "<collection>", "key": "<keyvalue>", "start": "<startvalue>", "end": "<endvalue>", "id": "<id>"}
  expected response: collection size, where the key "par" gives the number of data points that would be returned by a getRecordsInRange() request with the same arguments.

3.2.2 Responses from server to client:

- session control
  syntax: {"cmd": "SESSION", "par": "<specifier>", "status": "<status>", msg": "<message>", "id": "<id>"}
  where <specifier> is one of "LOGIN", "AUTH", oder "LOGOUT", <status> is either "OK" or "FAIL" and <message> is a string, e.g. an error message
  "OK" as status for "LOGIN" and "AUTH" indicates a successful login, i.e. that the client session has started. "OK" as status for "LOGOUT" indicates that the server had ended the session and the user has been logged out.

- list of collections
  syntax: {"cmd": "LIST_COLL", "par": ["<collection>"], "id": "<id>"}
  where <collection> is the name of a collection

- collection size
  syntax: {"cmd": "COLL_SIZE", "name": "<name>", "key", "", "par": "<size>", "id": "<id>"}
  syntax: {"cmd": "COLL_SIZE", "name": "<name>", "key", "<key>", "start": "<start>", "end": "<end>", "par": "<size>", "id": "<id>"}
  where <name> is the name of a data collection and <size> is the number of data points in this collection . If <key> is not an empty string, then <size> will be the number of data points from the given collection for which the key value is in the interval [<start>, <end>).

- single data point
  syntax: {"cmd": "DATASINGLE", "name": "<collection>", "idx": "<idx>", "data": "<data>", "id": "<id>"}
  where <collection> is the name of a collection, "idx" has either the value "start" or "end", and <data> is a string representing a single data point in JSON format.

- data endpoints
  syntax: {"cmd": "DATA_ENDPOINTS", "name": "<collection>", "data": ["<start>", "<end>"], "id": "<id>"}
  where <collection> is the name of a collection and <start> and <end> are the first and last data point of this collection.

- data set
  syntax: {"cmd": "DATA", "name": "<name>", "key": "", "data": [<record>], "id": "<id>"}
  syntax: {"cmd": "DATA", "name": "<name>", "key": "<key>", "start": "<start>", "end": "<end>", "data": [<record>], "id": "<id>"}
  where <name> is the name of a data collection and each record is a JSON object describing a data point from this collection. If <key> is not an empty string, then the data array will contain those data points from the given collection for which the key value is in the interval [<start>, <end>).

3.3 Back-End

This subsection deals with the back-end of the ADIN INSPECTOR. How the system deals with client http calls, and how kafka interacts with the system. An overview of the system can be seen in Figure 3.21. Smaller subsections have been expanded in Figure 3.18, Figure 3.19, Figure 3.20.

The connection to the client is handled in the class Hub which contains handlers for the network interface. This class uses a separate class (ClientProtocolHandler) to parse and handle requests from the handler. This setup is according to the strategy design pattern and allows easily modifying or even replacing the client server-protocol. The Hub class and the ClientProtocolHandler access the database via an object that implements the IUserSession interface and encapsulates the database session. Currently there is only an implementation for MongoDB access (MongoDBUserSession), but the abstraction via the IUserSession interface allows to add classes that offer access to Kafka or other databases. Classes that implement IUserSession are instantiated with a factory Method (UserSession()) which guarantees that the returned object represents a successfully logged in database session.

3.3.1 Class Diagram

The overview in Figure 3.21 shows a number of classes and their interaction with each other. What follows is a more in-depth view of what each component of this diagram does, what data it stores and how it fits into the overarching architecture.

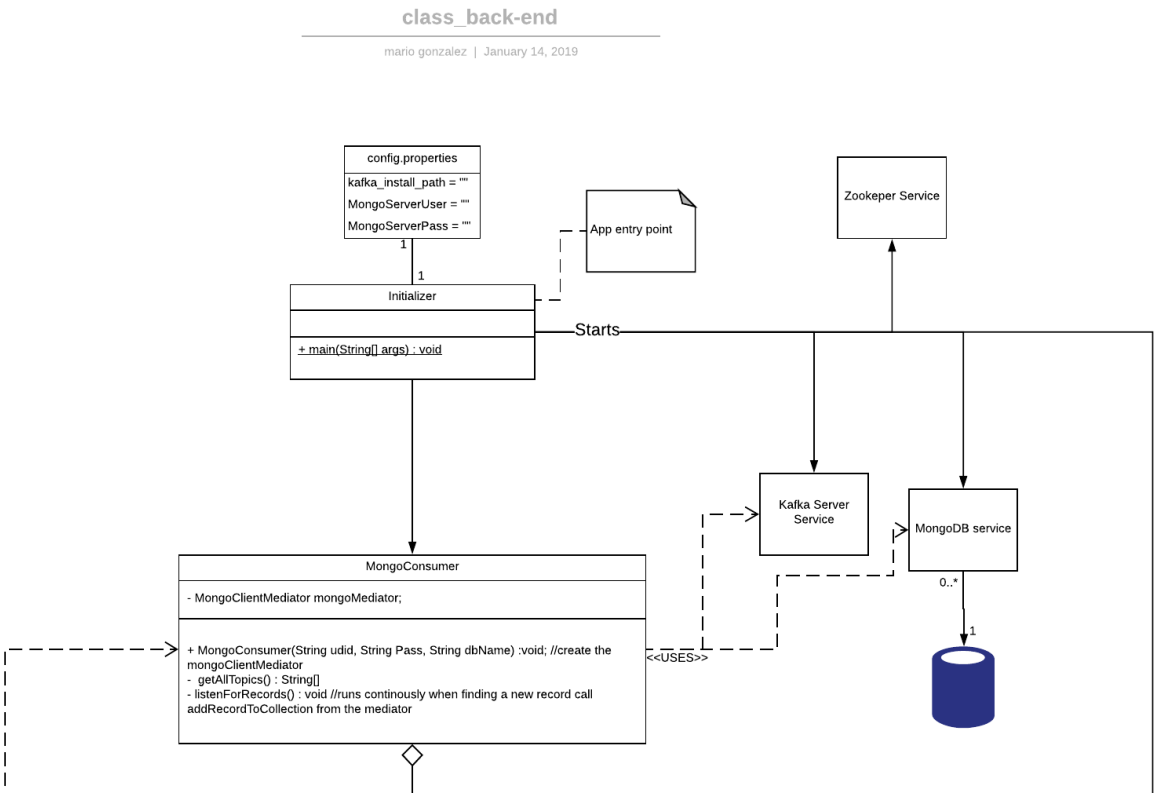


Figure 3.18: The classes involved in the initialization setup

- Config properties file
  The config file is stored alongside the built application .jar file and contains the path to the Kafka installation folder, the user name and password of a mongoDB account with the highest level of access and the name of the database.

- Initializer
  Methods:

  - main
    parameters: String of arguments from the console
    returns: void
    App entry point.
    We load the config.properties life and use the path provided to start the zookeeper, kafka and mongodb services

- MongoConsumer
  The Mongo Consumer, as the name implies, consumes all messages from all topics in the Kafka messaging system. Once a message is found it is passed along to the Mongo Client for further processing.
  Attributes

  - clientMediator
    Type : MongoClientMediator

An instance of the Mongo Client Mediator, created with the credentials from the config file.

Methods

- MongoConsumer constructor
  parameters: user name and password of a mongoDB account with the highest level of access.
  Initializes the MongoClient variable and calls listenForRecords();
- getAllTopics
  parameters: none
  returns: an array of strings containing all the available kafka Topics.
  Asks the kafka server service which topics exists.
- listenForRecords
  parameters: none
  returns: void
  This Method first calls getAllTopics and uses the array of topics to poll the kafka server for new messages.
  If new messages are found then the messages are passed to the Mongo Mediator for adding them to the Database.
  If no new messages are found for a topic notify the Mongo Mediator that the collection tied to the topic is ready for pre-processing.

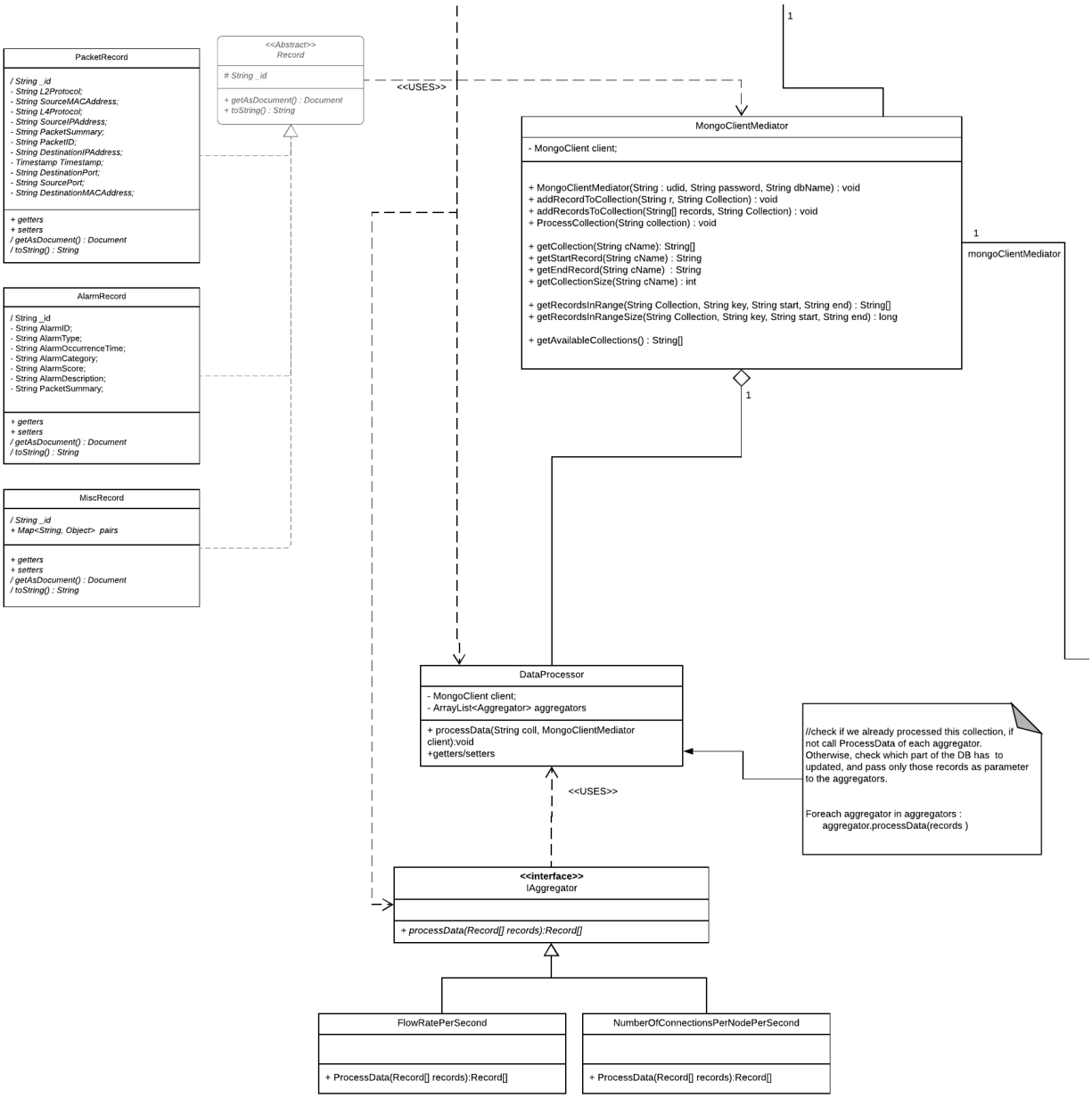


Figure 3.19: The classes involved in reading and writing data into the database

- MongoClientMediator This object serves as a nexus between the users who want to get data out of the database and the consumer and dataProcessor who want to add data into the database. This class encapsulates the mongo client from the mongo API. This means that any user wanting to sign in has to have valid credentials in the database, effectively relegating UAC to mongoDB.
  Attributes
  - client
    type: MongoClient
    An instance of the Mongo Client from the official java API.
  - dataProc
    A reference to the data processor class for this client.

  Methods
  - MongoClientMediator constructor
    parameters: Username and password
    Initializes the client variable, throws an error if the user is not found.
  - addRecordToCollection
    parameters: String representation of a record in json format
    String name of the collection it should be added to.
    returns: void
    Converts the json string into a java object, then to a bson document and uses the mongoAPI to insert it into the database.
  - addRecordsToCollection
    parameters: String Array of records to be added to a collection
    String name of the collection it should be added to.
    returns: void
    for each oneof the members of the array call addRecordToCollection
  - ProcessCollection
    parameters: String, name of a collection
    returns: void
    signal the data processor to start the processing of a collection
  - getCollection
    parameters: String, name of a collection
    returns: String array containing all entries of the collection
  - getStartRecord
    parameters: String, name of a collection
    returns: the first entry of the collection as a String.
  - getEndRecord
    parameters: String, name of a collection
    returns: the last entry of the collection as a String.

  - getCollectionSize
    parameters: String, name of a collection
    returns: the number of entries in the collection as int
  - getRecordsInRange
    parameters: String, name of the collection to query
    String, key of the parameter used for filtering
    String start and end ranges for the filtering
    returns: String array containing all entries of the collection within that range
    this Method is very general to allow for flexibility.For example by letting the key be, SourceIPaddresses, or a timeStamp.
  - getRecordsInRangeSize
    parameters: String, name of the collection to query
    String, key of the parameter used for filtering
    String start and end ranges for the filtering
    returns: number of elements matching the range as int
  - getAvailableCollections
    parameters: -
    returns: String array with collection names
    Returns an array with the names of the collections available to the current user.

- Record
  Every message that comes from kafka and needs to be added to the database has its own Record class that inherit from this one.
  Every single class that inherits needs to be able to, using reflection, convert itself into a Bson Document where every variable is a key Value pair of the name of the variable and its associated value.
  Attributes
  - id
    type: String

  Methods
  - getAsDocument()
    parameters: none
    returns: A Document, containing every variable of any class inheriting from this one.
    This function checks for every variable, gets its name and value as a string and adds it to the document that it eventually returns.

- PacketRecord
  Inheriting from Record, this class contains the variables that match the json string obtained from kafka.
  Attributes
  - id
    type: String

this id is used for determining the ordering when saving to mongoDB, it's the offset of the message in the kafka messaging queue. inherited from Record

- client
  type: String
- L2Protocol
  type: String
- SourceMACAddress
  type: String
- L3Protocol
  type: String
- L4Protocol
  type: String
- SourceIPAddress
  type: String
- PacketSummary
  type: String
- DestinationIPAddress
  type: String
- Timestamp
  type: String
- DestinationPort
  type: String
- SourcePort
  type: String
- DestinationMACAddress
  type: String

Methods

- getters / setters
  parameters: variable
  returns: variable type
  Each variable has its getters and setter methods.

- AlarmRecord
  Inheriting from Record, this class contains the variables that match the json string obtained from kafka.
  Attributes
  - id
    type: String
  - AlarmID
    type: String

- AlarmType
  type: String
- AlarmOccurrenceTime
  type: String
- AlarmCategory
  type: String
- AlarmScore
  type: String
- AlarmDescription
  type: String
- PacketSummary
  type: String

Methods

- getters / setters
  parameters: variable
  returns: variable type
  Each variable has its getters and setter methods.

- MiscRecord
  Inheriting from Record, this class is used by the data processor as an 'in-between' state before saving to the database. As well as an extension point for adding more types of records into the database programatically in the future.
  Refer to the data processor class for further data on the key value pairs.
  Attributes
  - pairs
    A Map of strings to Objects to store any 1 to many relationships

  Methods
  - getters / setters
    parameters: none
    returns: variable type
    Each variable has its getters and setter methods.

- DataProcessor
  This class is a mediator for each one of our data aggregators used for extraciton of features from the raw data stored in mongoDB.
  We might want to hve multiple data processors for chaining different aggregators together or to split up the work into mutliple threads. This is dependant on further performance testing.
  Attributes
  - client
    an instance of the associated mongoClient that requested the data aggregation
  - aggregators
    A Arraylist containing all the aggregators to be applied on a collection.

Methods

- getters / setters
- processData
  parameters: variable
  returns: variable type

- IAggregator
  This interface is the building block for every aggregator to be applied to data
  Attributes Methods
  - processData
    parameters: Records array of the records to be processed

- FlowRatePerSecond
  Implements IAggregator. This calculates, per port, the outgoing and incoming connections. A record processed by this aggregator is stored in a collection as follows:

```
  Name of collection: collectionName\_FlowratePerSec
  structure of record as json:
   {
  "date" : \{" date" " Unix_Timestamp  }
  rounded down to the second this record points to.
  Connections : [
  { Port: "portNumer", "InOut" : " In/Out ", count : "Number" }
  { Port: "portNumer", "InOut" : " In/Out ", count : "Number" }
  ...

  ] This array has an entry per port if the port communicated that second.
  Precomputing this allows us to stream whenever the client needs the information
  for a specific node.
}
```

  Methods

  - processData
    parameters: Records array of the records to be processed
    specific imlpementation left to the classes implementing this interface

- NumberOfConnectionsPerNodePerSecond
  Implements IAggregator. This calculates the outgoing and incoming connections. A record processed by this aggregator is stored in a collection as follows:

```
  Name of collection: collectionName\_FlowratePerSec
  structure of record as json:
   {
  "date" : \{" date" " Unix_Timestamp  }
  rounded down to the second this record points to.
```

```
  Connections : [
  { Port: "portNumer", count : "Number" }
  { Port: "portNumer", count : "Number" }
  ...

  ] This array has an entry per port if the port communicated that second.
  Precomputing this allows us to stream whenever the client needs the information
  for a specific node.
}
```

Methods

- processData
  parameters: Records array of the records to be processed

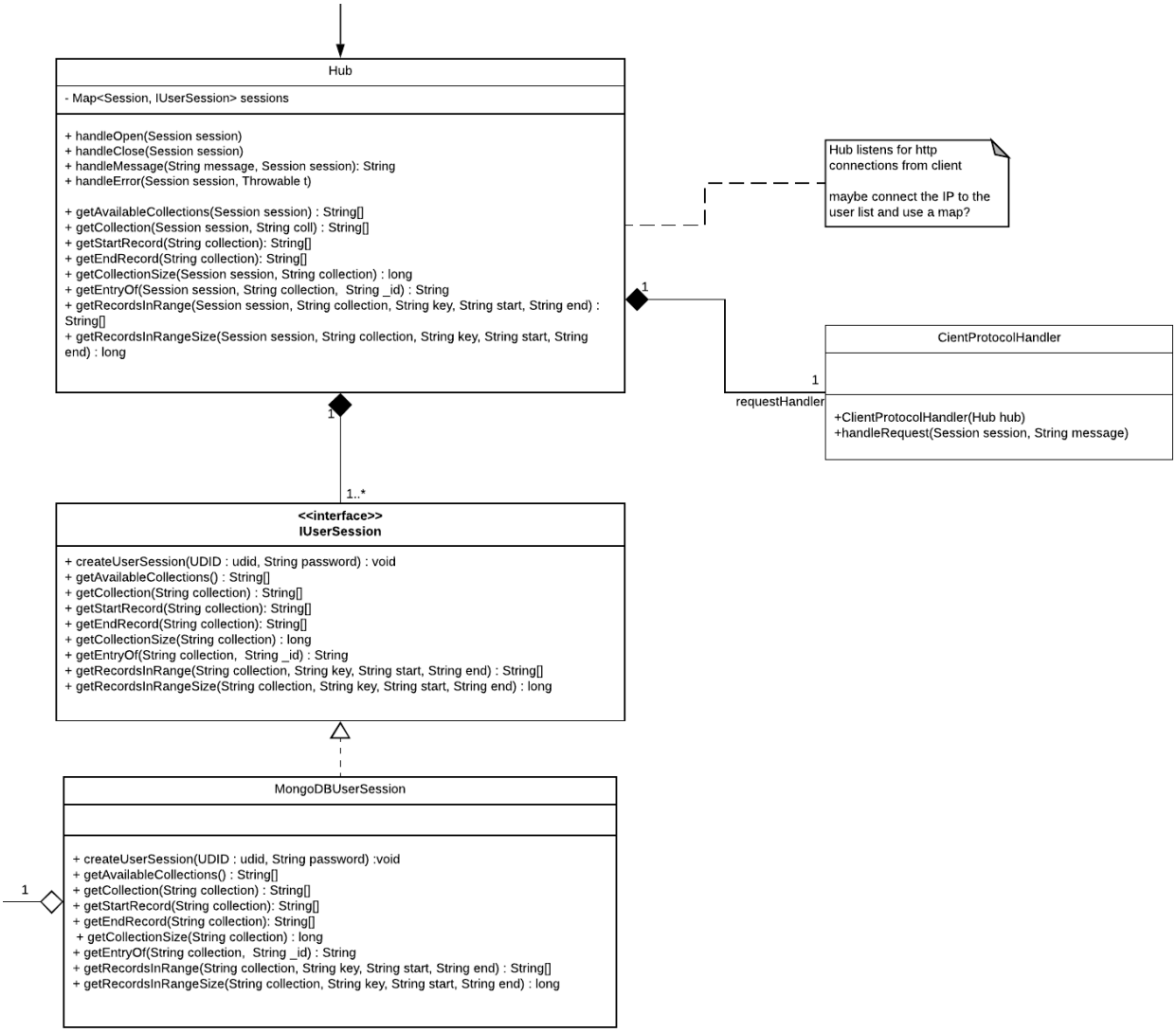


Figure 3.20: The classes involved in the communication between the server and the client

- Hub
  This class implements the network handlers for the WebSocket connection to the client. It also has wrapper methods that delegate the database commands to the appropriate IUserSession object.
  Attributes
  - requestHandler
    Type : ClientProtocolHandler
    The strategy object we call for the actual parsing of the client requests.
  - sessions
    Type : map<Session, IUserSession>
    A map that connects a client communication session to a database session. A non-null entry represents a successfully logged in user session.

  Methods
  - handleOpen
    parameters: Session session - the current session
    returns: void
    Event handler for the start of WebSocket connection.
  - handleClose
    parameters: Session session - the current session
    returns: void
    Event handler for closing a connection.
  - handleMessage
    parameters: String message - the message that we received from the client
    Session session - the current session
    returns: String - the response to be sent to the client
    Event handler for receiving a message. The message is passed to the ClientProtocolHandler.
  - handleError
    parameters: Session session - the current session
    Throwable t - the exception that occurred
    returns: void
    Event handler for errors/exceptions during communication.
  - createUserSession
    parameters: Session session - the current session
    String username - the user id to login with
    String password - the password
    returns: IUserSession
    Delegate to the IUserSession method to instantiate a new UserSession and log in into the database using the given credentials.
  - getAvailableCollections
    parameters: Session session - the current session

returns: String array with collection names
Returns an array with the names of the collections available to the current user.

- getCollection
  parameters: Session session - the current session
  String - name of a collection
  returns: String array containing all entries of the collection
- getStartRecord
  parameters: String, name of a collection
  returns: the first entry of the collection as a String.
- getEndRecord
  parameters: String, name of a collection
  returns: the last entry of the collection as a String.
- getCollectionSize
  parameters: Session session - the current session
  String collection - the collection to query
  returns: long - the number of records
  Returns the number of records in the specified collection.
- getRecordsInRange
  parameters: Session session - the current session
  String - name of the collection to query
  String key - the parameter used for filtering
  String start and end - range for the filtering
  returns: String array containing all entries of the collection within the filter range
  Returns an array containing all records of this collection for which the value of the specified key is in the range [start, end). The records will be in the same order as they are in the collection.
- getRecordsInRangeSize
  parameters: Session session - the current session
  String - name of the collection to query
  String key - the parameter used for filtering
  String start and end - range for the filtering
  returns: number of elements matching the range as int
  Returns the number of records in the specified collection for which the value of the specified key is within the range [start, end).

- IUserSession
  An IUserSession object encapsulates a data base session. On instantiation an IUserSession connects to a database using the given user id and password and uses this connection for all following data base access.
  Methods
  - createUserSession
    parameters: String username - the user id to login with
    String password - the password
    returns: IUserSession
    Factory method to instantiate a new UserSession and log in into the database using the given credentials.
  - getAvailableCollections
    parameters: -
    returns: String array with collection names
    Returns an array with the names of the collections available to the current user.
  - getCollection
    parameters: String - name of a collection
    returns: String array containing all entries of the collection
  - getStartRecord
    parameters: String, name of a collection
    returns: the first entry of the collection as a String.
  - getEndRecord
    parameters: String, name of a collection
    returns: the last entry of the collection as a String.
  - getCollectionSize
    parameters: String collection - the collection to query
    returns: long - the number of records
    Returns the number of records in the specified collection.
  - getRecordsInRange
    parameters: String - name of the collection to query
    String key - the parameter used for filtering
    String start and end - range for the filtering
    returns: String array containing all entries of the collection within the filter range
    Returns an array containing all records of this collection for which the value of the specified key is in the range [start, end). The records will be in the same order as they are in the collection.
  - getRecordsInRangeSize
    parameters: String - name of the collection to query
    String key - the parameter used for filtering
    String start and end - range for the filtering
    returns: number of elements matching the range as int

Returns the number of records in the specified collection for which the value of the specified key is within the range [start, end).

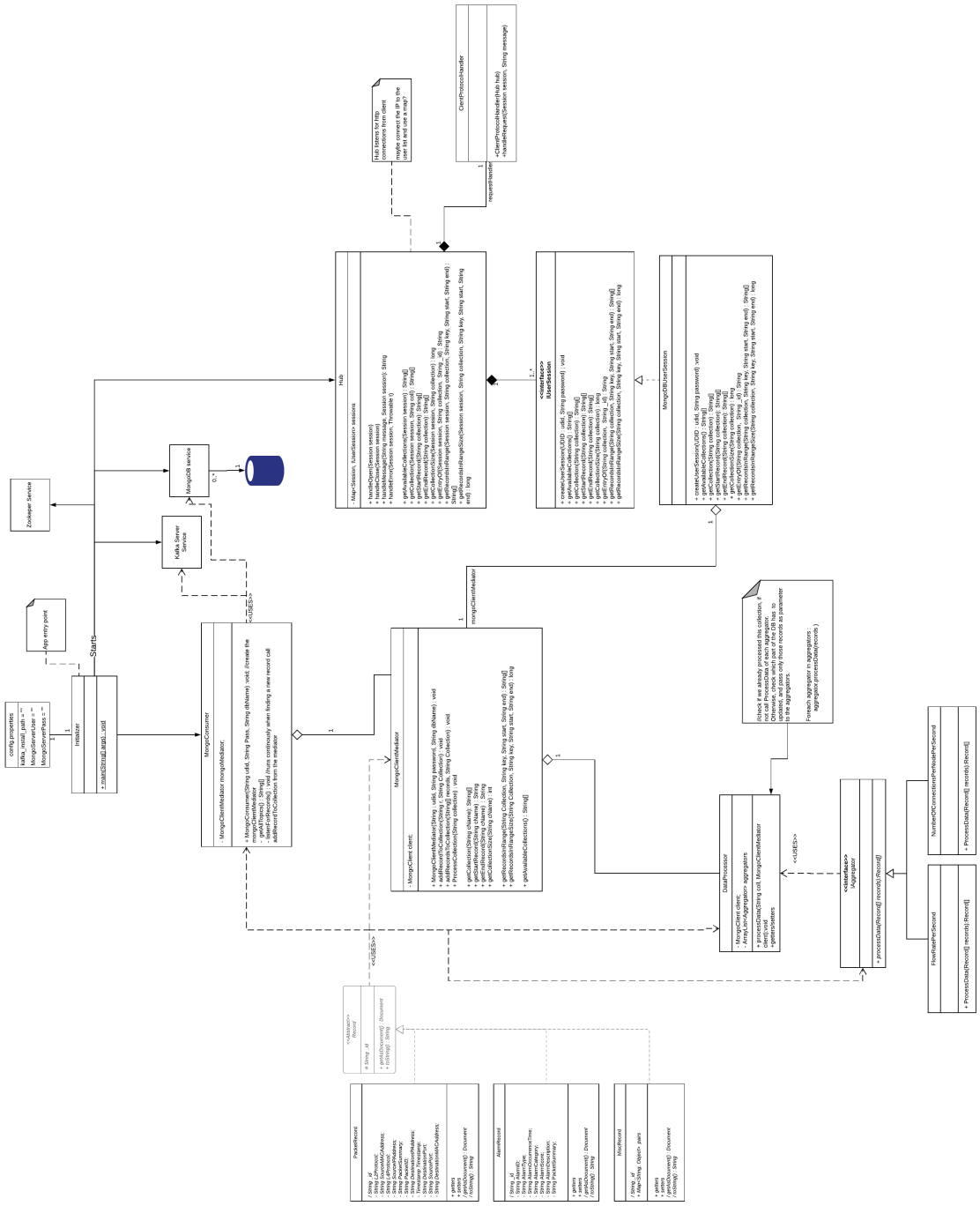
- MongoDBUserSession
  Encapsulates a user session for a connection to a MongoDB database.
  Attributes
  - mongoClientMediator
    Type : mongoClientMediator
    The mediator object used to access the database

  Methods
  - MongoDBUserSession constructor
    parameters: -
    Private constructor to create a new MongoDB session.
  - createUserSession
    parameters: String username - the user id to login with
    String password - the password
    returns: a new MongoDBUserSession object
    Factory method to instantiate a new MongoDBUserSession and log in into the database using the given credentials.
  - getAvailableCollections
    parameters: -
    returns: String array with collection names
    Returns an array with the names of the collections available to the current user.
  - getCollection
    parameters: String - name of a collection
    returns: String array containing all entries of the collection
  - getStartRecord
    parameters: String, name of a collection
    returns: the first entry of the collection as a String.
  - getEndRecord
    parameters: String, name of a collection
    returns: the last entry of the collection as a String.
  - getCollectionSize
    parameters: String collection - the collection to query
    returns: long - the number of records
    Returns the number of records in the specified collection.
  - getRecordsInRange
    parameters: String - name of the collection to query
    String key - the parameter used for filtering
    String start and end - range for the filtering
    returns: String array containing all entries of the collection within the filter range
    Returns an array containing all records of this collection for which the value of the specified key is in the range [start, end). The records will be in the same order as they are in the collection.

- getRecordsInRangeSize
  parameters: String - name of the collection to query
  String key - the parameter used for filtering
  String start and end - range for the filtering
  returns: number of elements matching the range as int
  Returns the number of records in the specified collection for which the value of the specified key is within the range [start, end).

- ClientRequestHandler
  This class handles client requests by parsing them, executing the requested action and producing responses. The requested actions are typically executed by calls to the appropriate methods in the Hub object. The relation between the Hub class and this class is basically the strategy design pattern with a single strategy.
  Attributes
  - hub
    Type : Hub
    The Hub object to work with

  Methods
  - ClientRequestHandler
    parameter: Hub hub - the Hub object to work with
    The constructor; sets the hub attribute.
  - handleRequest
    parameters: Session session - the current client session
    String message - the client request to process
    returns: String - the response to be sent to the client
    Parse the message from the client, execute the requested action, and construct the response message.


class back-end
Hub
ClientProtocolHandler
MongoDbUserSession
MongoConsumer
MongoClientMediator
DataProcessor
FiveYearPerSecond
NumberOfConnectionsPerNodePerSecond
PulseRecord
AlarmRecord
MiscRecord
Starts
Initializer
App entry point
Zookeper Service
Kafka Server Service
MongoDB service

3.3.2 Sequence Diagrams

Figure 3.22 shows the initalization sequence order, the correspoding class diagram is Figure 3.18, the program dependes on a couple of services namely (in order), the zookeeper serivce, the kafka server service and the mongoDB service. Once all services are up and running the MongoConsumer is created and can start consuming messages and the Hub can start listening to client logins, requests, etc.

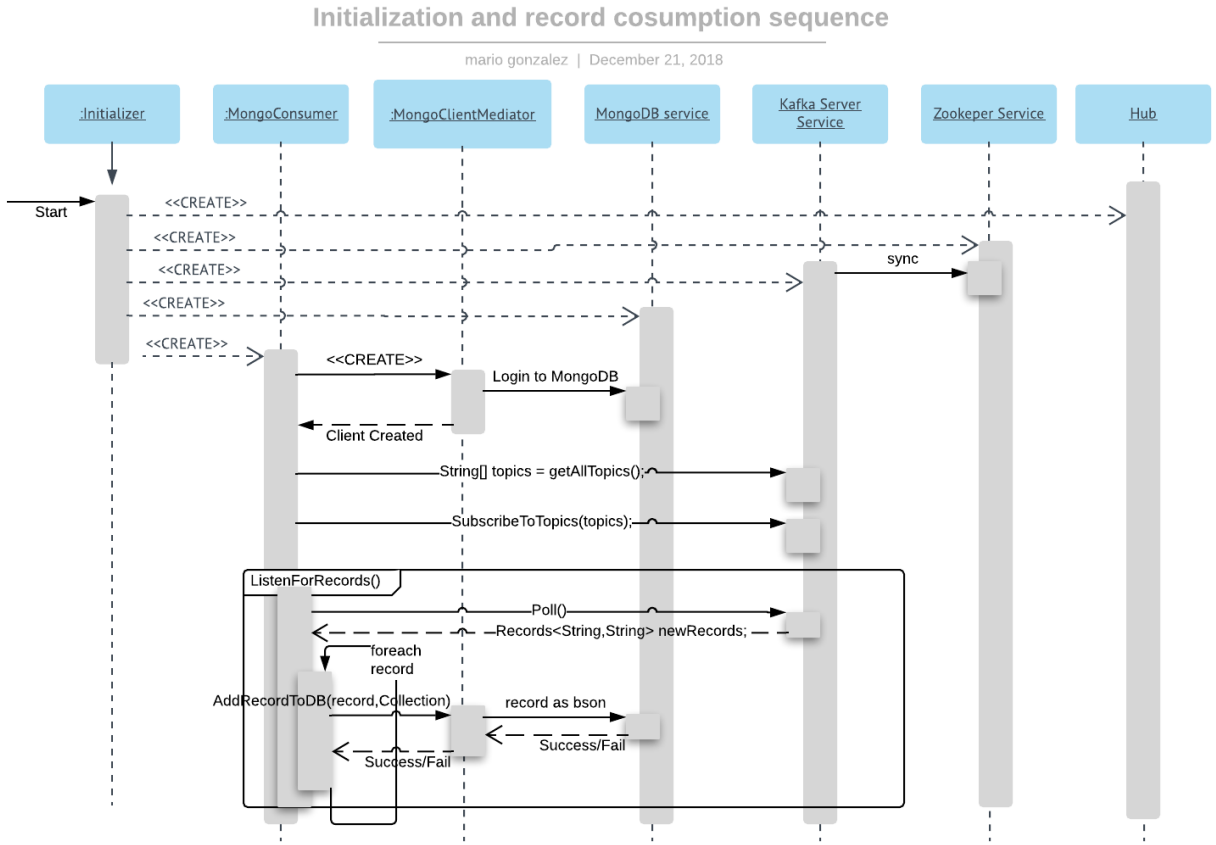


Figure 3.22: Initialization sequence and message consumption

3.3.3 Activity Diagram

As previously mentioned for UAC the built-in UAC in MongoDB is used.
Figure 3.25 shows the workflow on adding new roles and new user, who upon creation have a role assigned to them, to MongoDB.
A Role determines what can be done and seen within a database. For the purposes of the ADIN INSPECTOR there are three basic roles, Admin, Operator and Analyst. The admin role , created by default, can create and destroy users as well as assign specific roles to them. An analyst can see all collections on the database and an Operator can only see part of them.

Comsuming messages {{lastModifiedBy}} | {{lastModifiedTime:MMMM d, yyyy}}

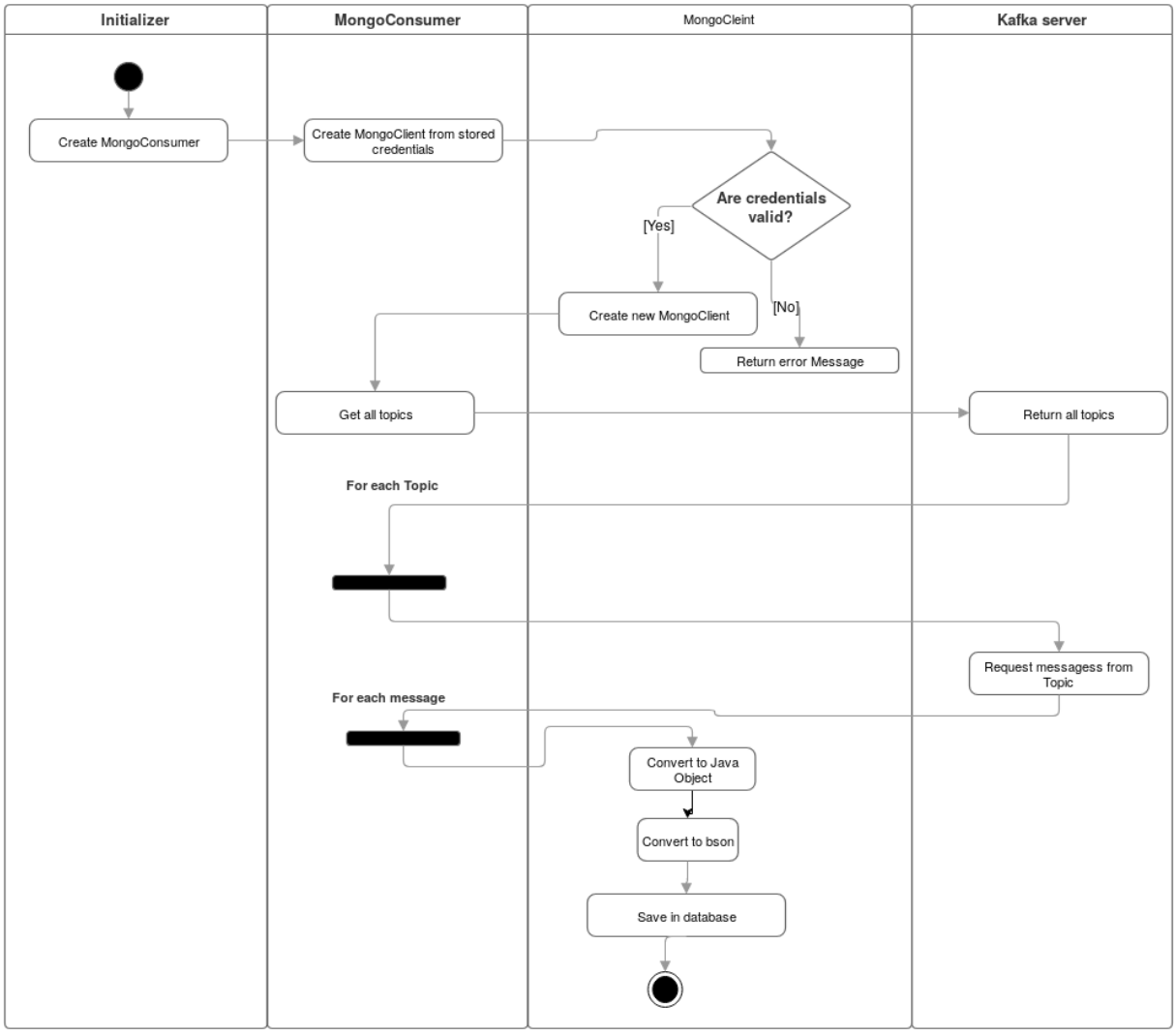


Figure 3.23: Initialization and message consumption workflow.

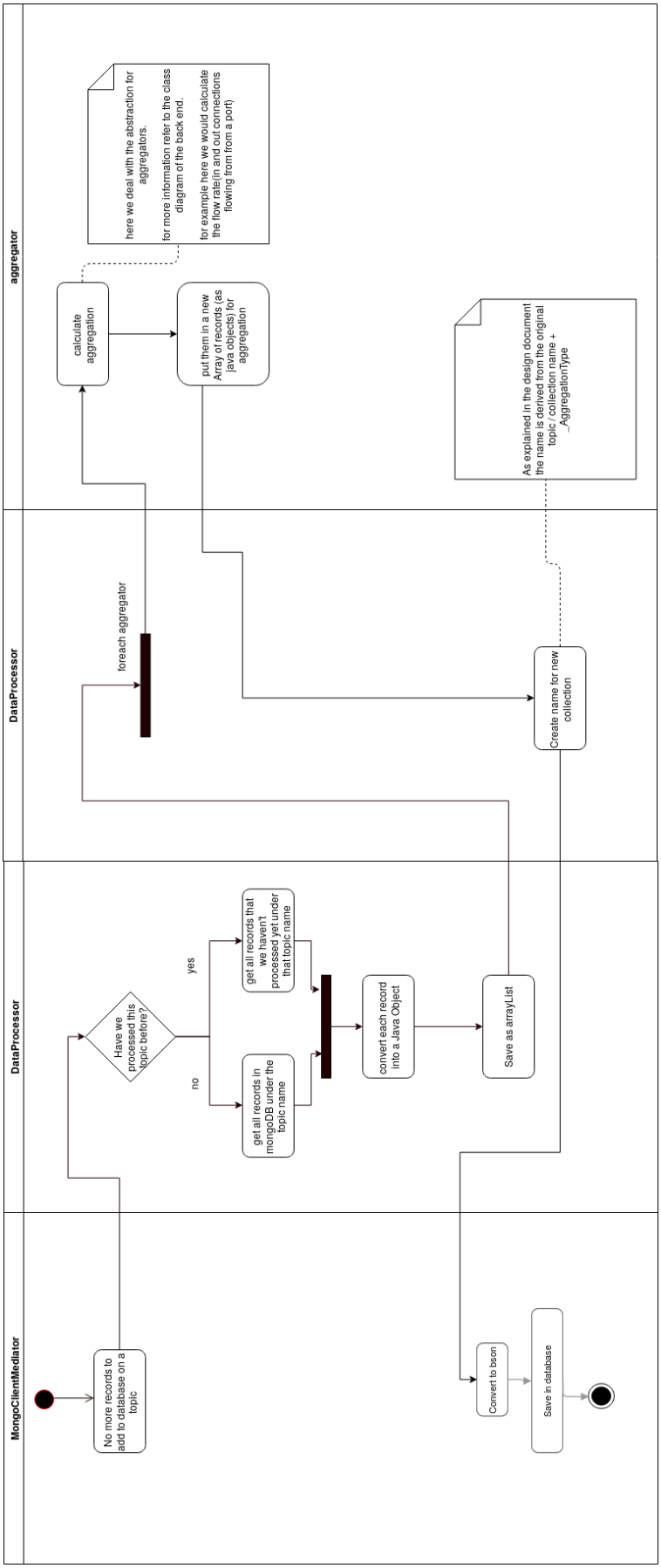


Figure 3.24: This diagram shows the processing of Collections and records as well as the addition of extracted data back into the database.

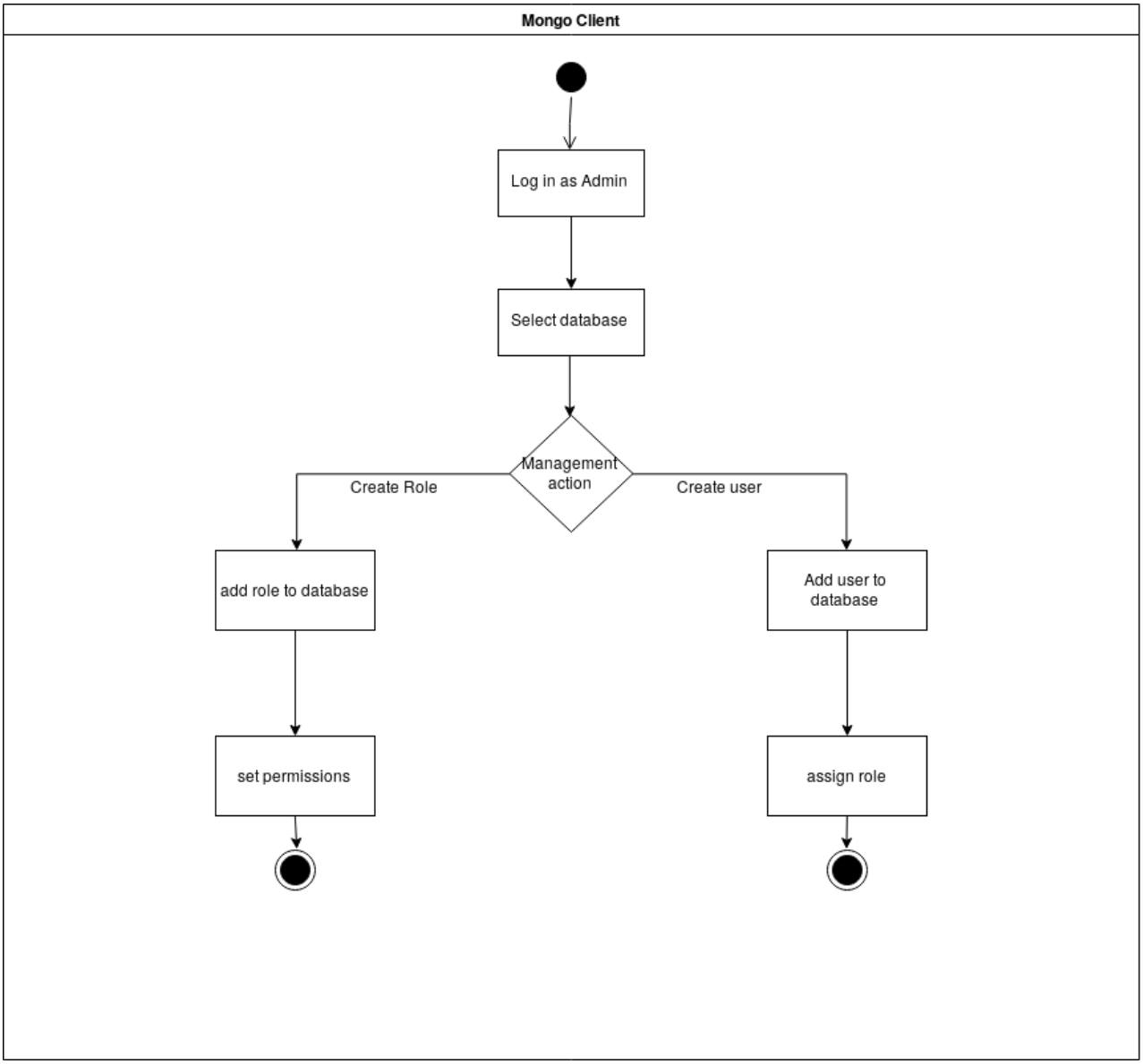


Figure 3.25: User Management workflow

3.4 Changes

Changes made between version 1.1.0 and version 2.0.0 of this document.

3.4.1 Changes to the client-server protocol

The protocol responses have been made more expressive in general.

- Rename "LOGIN_TOKEN" request to "AUTH".
- Rename "LIST_COL" response to "LIST_COLL".
- Expand "session" response with the specifier key "par"
- Expand "data set" and "collection size" responses with the key "key"
- Expand "data set" and "collection size" responses with the keys "start" and "end", if they were in the request
- "data set" and "collection size" responses will now echo back any key-value pairs not specified in the protocol
- getStartRecord, getEndRecord, and the "single data point" response were added. itemgetEndpoints and the "data endpoints" response were added. end XXX

4 Implementation

4.1 Introduction

This implementation report covers changes in the design from the one decribed in the original design phase and decribes the current state of the implementation by lists of completed and not completed requirements.

4.2 Changes in the Design

Changes are grouped together according to the reason for the change.

4.2.1 User Interface changes for better aesthetics and convenience

- New Login Page
  Opted for a different graphical design than the one in the mockups due to aesthetic reasons. The functionality and behavior of the login page remains the same as in the design docs.
- Removal of the filter button inside diagram control containers
  The filters now can be easily accessible inside the config modal of the diagram.

4.2.2 User Interface changes for usability improvements

- New text input box for WebSocket endpoint
  Added the feature of choosing an arbitrary WebSocket endpoint by persisting the input with Browser Local Storage API, making the frontend application completely standalone and therefore largely simplified the deploy process of the entire DHSTTOS suite.

4.2.3 Refactoring for cleaner code and changes for convenience reasons

- Data formatting helper function on the frontend
  Added a helper function **formatData = ({ groupName, x, y, rawData = [] }) : Object[]** which converts the raw data points the frontend receives from the server to structured data arrays which allows easy data passing into the diagram drawing routines.
- Add parameter
  Added parameter DBname to **MongoConsumer(user, pass, dbName)** for creating a reference to pass onto the MongoClientMediator

- Refactoring
  Add attribute **private KafkaConsumer<String, String> consumer** because other functions need to use the consumer

- Refactor: extract instance attribute
  Add attribute **private MongoDatabase db** as a reference to the database all methods need to access.

- Convenience functions for different data types
  Added variations of **addRecordToCollection(Record record, String collection)** that take a document or an list of documents or an array of record sinstead of a Record.

- Add convenience function
  Added **getCollectionAsRecordsArrayList()** to DataProcessor.

- Refactor passing the current mediator object
  Add parameter **MongoClientMediator** to **public static void ProcessData:processData(String collectionName, MongoClientMediator clientMediator)** so that **processData** can use it to write the processed data to the database. Remove attribute **ProcessData:MongoClientMediator client** which was used for this before.

- Add convenience function
  Add method **public static void processData(ArrayList<String> collectionNames, MongoClientMediator clientMediator)** to process a list of collections (instead of calling **processData** for each collection.

- Add convenience function
  Added method **public Document getNewAggregatorDocument(Date tstmp)** for easier handling of date values.

- Add convenience attributes
  Add the variables Variables **private ArrayList<Map<String, Object» connectionsMapList** and **private Document currentDocument** to the classes **FlowRatePerSecond** and **NumberOfConnectionsPerNode** to keep track of which document is being processed now and which connections happened within this second.

- Refactoring for cleaner code in protocol handling
  Change the protocol parsing in class **ClientProtocoHandler** from a switch construct to using a private enum.

4.2.4 Changes due to clarified requirements

- Differing input formats for Date/Timestamp
  Split class **PacketRecord** into **PacketRecordDesFromMongo** and **PacketRecordDesFromKafka** to handle different formats.

4.2.5 Changes due to oversights

These are changes and additions due to oversights and mistakes in the original design.

- added dbName to MongoClientMediator since we need to know from which DB we want to read/write collections.

- Unspecified return type
  The return type of **public ArrayList<Document> processData(ArrayList<Record> records)** in **IAggregator** was unspecified in the Design document.

- Session handling
  To handle session state, **Hub:login()**, **Hub:loginWithToken()**, and **Hub:logout()** were added. To keep track of client session state, the private attributes **Hub:sessions** and **Hub:loginTokens** were added.

4.2.6 Changes due to unexpected complexity

These changes can be attributed to lack of familiarity with the used components and libraries.

- Workaround for Kafka's API
  Change **getAllTopics()** to **getAllTopicsPartitions()**: return a Collection of topic partitions essentially to force kafka to send all records from the start. It was complex to make kafka read all the topics from the beginning. Secondary aspect: convenient because it relegates topic creation to another method.

- Workaround for Kafka's API
  Add method **ArrayList<String> getTopicsForProcessing()** because there are some topics in kakfka which are for internal use, e.g. __consumeroffsets. This returns the topics we need to process.

- Exception handling
  The constructor for class **MongoClientMediator** now throws a LoginFailureException instead of forwarding an unchecked exception.

- Converting between different APIs
  Add method **mongoIteratorToStringArray(MongoIterable)** because the hub expects an array but the mongodb returns a MongoIterable.

- Making the network protocol more expressive to simplify handling responses

- Handling the login happening in another websocket session than the main app

  To deal with a restart of the websocket connection when changing from the login page to the main page, session handling was changed. Added the **LOGIN_TOKEN** request to the protocol and **Hub:loginWithToken**. Also, the client can add new key-value-pairs to the request which will be returned unchanged by the server.

- Adapt to React and MobX
  To adapt to the observer-driven architecture of React and MobX, store data from the server in datastructures **dataStore.rawData** and **dataStore.alarms** instead of returning it as return values of **getAvailableCollections()**, **getCollection()**, **getCollectionSize()**, **getRecordsInRange()** and **getRecordsInRangeSize()** in **wsutils.js**.

4.3 List of implemented must- and should-requirements

4.3.1 List of implemented must-requirements

FR100, FR110, FR200, FR300, FR400, FR500, FR700, FR710, FR720, FR1110, FR1300, FR1310
cancelled after Mike left: FR800

4.3.2 List of implemented should-requirements

- FR1332 filter to compute flow rate
  - this has instead been implemented in the backend which provides this as a new data stream
- FR1400

4.3.3 List of not implemented must-requirements

- FR600 dynamically change the selected/displayed components
- FR900 The amount of data can be limited via a slider [...] to which all diagrams must update to.
- FR910 Within the slider the user is able to scroll through the timeline and the diagrams need to react in real-time.
- FR1000 auto scroll
- FR1100 pick data points, hover
- FR1200 selecting data points
- FR1210 create new diagram from selected data
- FR1330

4.3.4 List of not implemented should-requirements

- FR1320 per-diagram filters

4.4 Timeline and Delays

There were delays in the implementation phase caused by

- clarification of requirements
- evaluation of graphics libraries
- familiarization with the used APIs and libraries

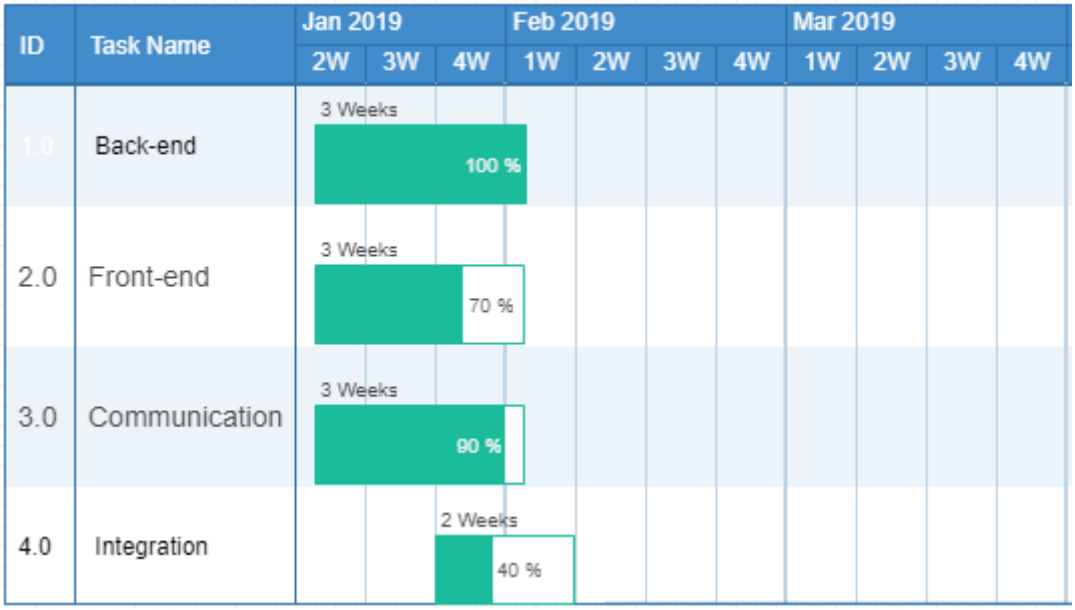


Figure 4.1: The timeline.

4.5 Overview of unit tests

Number of unit tests:

DataProcessor Test 1 and 2 both test the functionality of trying to process missing data. In both cases the aggregators responsible should not fail and simply make sure nothing gets added into the Database.	2
Hub Test session handling methods.	4
MockMongoDBUserSession	4
MongoClientMediator Test1 and 2 and 3: test different credential errors, like missing credentials or credential without access to the necessary databases. All tests cases return a LoginFailureException. Test 4 and 5 test handling null reference objects to adding records into MongoDB. In this case then simply, nothing is added into the database.	5
MongoConsumer Test 1 : tests the case on failure to communicate with kafka, the consumer exists and reports this to the console but operation of all other parts of the program continue since failure of this does not affect working with data already stored in the database. Test2: tests case when the consumer does not have the right credentials to access the MongoDB, the Consumer should then fail to be created and call a LoginFailureException.	2
TestClientProtocolHandler	2
FormatData. Testing the GUI would be possible however, this is both complex and time consuming. Due to time constraints minimal testing was done outside of basic functionality.	

5 Testing

5.0.1 Introduction

This testing report covers changes and bug-fixes from the one described in the implementation phase and describes the current state of the project.

5.0.2 Issues Resolved

5.0.2.1 Tooltip on Node-Link diagram not working

See GitHub issue #69.

- Symptom
  Tooltips don't show up somehow.
- Solution
  Heavily modified tooltips code, and applied correct CSS styles to correctly render the tooltips.

5.0.2.2 Data Store updates break everything

Also see GitHub issue #69.

- Symptom
  Calling myNetwork.updateData within the callback passed to MobX autorun function crashed everything.
- Solution
  The reason was that hidden deep inside the update() function of the node-link diagram, there were some "functions" that mutate the passed in parameters themselves, adding in D3 positioning info and modifying other stuff - so in our case it's the data object inside dataStore gets mutated and it becomes useless when it gets passed inside myNetwork.updateData(), which is why it all chokes when MobX runs myNetwork.updateData() later on.

5.0.2.3 Incompatibility issues of data pulling

See discussions under GitHub issue #69 and pull request #78.

- Symptom
  Due to some issues, the frontend-backend communication was not working properly. Data pulling was not working.
- Solution
  Fixed the incompatible API calls on both server and client side.

5.0.2.4 Filter and grouping features not working

See discussions under GitHub issue #86, #74 and pull request #101.

- Symptom
  The filter and grouping features were not working due to an incomplete implementation.
- Solution
  Completed the feature.

5.0.3 Bug fixes

This section lists bugs which were found and fixed in the testing phase that were not recorded as GitHub issues.

5.0.3.1 Real-time data being persistent

When listening for Real-time data there was not limit to how much of a buffer of previous sessions.

- Symptom
  Closing and restarting the program at a latter time would keep the real-time collection when the two distinct times should not be concatenated together on the same collection.
- Solution
  When the program first starts, discard all real-time date pertaining to other sessions and creating a buffer from there.

5.0.3.2 Real-time data buffer keeps growing

when listening for Real-time data there was not limit to how much of a buffer of the past

- Symptom
  Once we started listening to real-time there was no limit to how big the buffer grew. As it grew queries become slower and more unnecessary data was stored.
- Solution
  Limit the size of the real-time collection to 60K entries. Roughly 10 minutes of buffer at 100 entries per second.

5.0.3.3 Real-time aggregated collections not being updated

Real-time aggregations were not being updated since the java API does not overwrite entries on MongoDB by default.

- Symptom
  Real-time aggregated collections are not updated.
- Solution
  Iterate over all aggregated collections and delete them on start alongside the real-time collection.

5.0.3.4 Real-time aggregated collections not visible from front end

- Symptom
  Real-time collections and aggregations do not show on the front-end. Since they do not exist if the program is not listening to the real-time Kafka topic.

- Solution
  On start, if not existing, create empty collections for real-time and it's aggregations to expose them to the front end.

5.0.3.5 Getting collections records between a range

- Symptom
  The range was inclusive at the start and inclusive at the end when it should be inclusive at start and exclusive at the end.

- Solution
  Changing the query to MongoDB from 'less or equal than' to 'less than'.

5.0.3.6 Getting collections records between a range when using _id

- Symptom
  The _id corresponds to the index of the collection. Getting for example the records from 0 as start to end 10 does not return records 0-9

- Solution
  Changed _id from a String to a long in the collection to allow for numerical comparisons.

5.0.3.7 collections are too large to effectively draw the node-link diagram

with really big data sets it's problematic to iterate over the whole thing to draw the diagram.

- Symptom
  Hang ups and freezes until the diagram is drawn and continuous freezes on redraws.

- Solution
  Created a new aggregation containing the information needed to draw the diagram including all connections over a data set as well as the ability to create subsets spanning a set range.

5.0.3.8 Handling of dates from Kafka and MongoDB

When converting an entry from Kafka to JSON containing a date the value it is converted to $date : long but bson doesn't automatically convert this to a Date Java object.

- Symptom
  Automatic failure of the bson parsing routine

- Solution
  Several revisions of this.
  First fix was ignoring the $date object altogether and create a mock Timestamp class holding the real value. while this fixes the issue it introduces another one where getting an entry from the MongoDB and using bson to convert it into a Java object does not work since the conversion does not hold this mock Timestamp class.
  Second fix was writing a deserialization routine and creating a two classes, one containing

the mock object and one without it. Fixes the issue but under testing it was found unreliable and requiring way too big of an overhead.
Final fix: Create a proper deserialization routine converting the portion of the record containing the date into it's own JSON object and extracting the value of $date as key fixes the issue when converting from Kafka. Converting from MongoDB skips the routine and works as intended

5.0.3.9 Fix ConcurrentModificationException in logout().

Insufficient knowledge of API: it is not safe to remove an element directly from a HashMap, while iterating over the map.

- Symptom
  ConcurrentModificationException was thrown.
- Solution
  Delete the element (a terminated user session) with the iterator's remove() method.

5.0.3.10 In MongoDBUserSession, catch MongoSocketOpenException.

A timeout during login, e.g. because the MongoDB server was not running, caused an exception that was not caught and handled.

- Symptom
  The Hub crashed when MongoDB was not running.
- Solution
  Catch and handle MongoSocketOpenException and rethrow it as LoginFailureException.

5.0.3.11 getCollectionGroup() needs collection name as parameter.

Forgot to add a parameter for the collection to be used to function getCollectionGroup(); instead it used a default value.

- Symptom
  Data from a wrong collection was returned.
- Solution
  Add name parameter.

5.0.3.12 Fix client sessions clobbering each other.

The java WebSocket implementation instantiates a new Hub object for each WebSocket connection that gets opened (started) by a client. Since the Hub classes constructor initialized the list of sessions, this would remove all currently existing sessions.

- Symptom
  When a client connected to the server, all current sessions of other clients were aborted.

- Solution
  Move the initialization code for the sessions map from the constructor to a static block.

5.0.3.13 Wrong handling of the data structure of a message in wsutils.

The data received from the server was unpacked incorrectly in handleDataGroup(); instead of a map-of-maps it is a map of JSON-strings.

- Symptom
  The diagram drawing routines threw an exception on accessing the extracted data and displayed a blank screen.
- Solution
  Fix access to the data structure by parsing the JSON strings.

5.0.3.14 Fix brush container layout.

The width of the React container for the brush and the height of the brush's area were wrong.

- Symptom
  The "add new diagram" button was pushed too far to the right and the brush's axis was too close to the container's border.
- Solution
  Reduce the container width and increase the axis' vertical offset.

5.0.3.15 Update tick labels when brushing.

The array index calculation to get the values for the tick labels was wrong.

- Symptom
  The movable axis had no tick labels.
- Solution
  Fix the calculation of the index offset.

5.0.3.16 Data selection with the brush for the node-link diagram

The node-link diagram uses data preprocessed by the server. Therefore brushing and zooming can't be done by the client by simply extracting sections of the data set.

- Symptom
  The node-link diagram didn't react to the brush or went blank.
- Solution
  Request node-link data for the currently selected time range from the server.

5.0.4 Test cases

T100 Successful login	pass
T101 Failed login	fail (password check is disabled to facilitate other tests)
T200 Open second diagram	pass
T210 Open multiple diagrams	pass
T220 Configure a diagram	pass in changed form
T300 Filtering	pass
T310 Filtering chaining	N/A, dropped requirement
T400 Full screen a diagram	pass
T450 Exit full screen	pass
T500 Play button	N/A, dropped requirement
T510 Time window	N/A, dropped requirement
T600 Investigate a data point	pass
T610 Investigate nodes and links	pass
T620 Select data points	fail
T630 Create diagram from selection	fail

5.0.5 Remaining Issues

- Handle logged-out state better