

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA  
Faculty of Informatics and Information Technologies  
Ilkovičova 2, 842 16 Bratislava 4

# Product documentation

Milestone 2 – Summer semester

Team project  
2018/2019

Team č. 04 : Fastar

Bc. Michaela Balážová  
Bc. Kamil Janeček  
Bc. Tomáš Jendrejčák  
Bc. Matúš Kalafut  
Bc. Matej Končál  
Bc. Michal Maňak  
Bc. Ján Vnenčák

Project leader: Ing. Kamil Burda

Date of last edit: 10.05.2019

## Table of contents

1. Introduction.....	3
1.1. Dictionary .....	3
2. Big picture.....	4
2.1. Description of the project.....	4
2.2. Our goals in the end of winter semester: .....	4
2.3. Our goals in the end of summer semester:.....	4
2.4. Review of goals for winter and summer semester .....	5
2.5. The view of components connection in the system .....	5
Device .....	5
Application.....	5
Database .....	6
Web interface .....	6
3. Modules of the system .....	7
3.1. Server module .....	7
3.2. Logger module.....	7
3.3. Machine learning module .....	8
4. Manuals.....	9
4.1. Manual for product deployment.....	9
4.1.1. Server .....	9
4.1.2. Logger-web.....	11
5. User testing .....	14
6. Technical documentation.....	15

# 1. Introduction

This document describes the software product, big picture of the project and its created modules. It also contains manuals and technical documentation to the project.

## 1.1. Dictionary

- **raw data** - data obtained directly from device
- **feature** - measurable property or characteristic of data
- **user model** - collection of characteristics describing user

## 2. Big picture

### 2.1. Description of the project

Computers and mobile devices are sources of sensitive data such as payments, mails or social networks. The theft of device or user account can have serious consequences. That is why we are working on authentication through behavioral biometrics. Its advantage over existing approaches (strong passwords, fingerprints) is the possibility of additional background authentication without user noticing.

Our project deals with behavioral biometrics and provides user authentication directly while using the device. Each person uses their device differently, whether they interact with a mobile device or a computer. Based on this theory and user's unique behavior, we are able to determine whether it is an authorised person to use the device and to prevent unauthorised access by a foreign person to the device.

### 2.2. Our goals in the end of winter semester:

- have a functional logger for a computer mouse and mobile devices
  - log various types of events
  - support multiple web browsers
- divide data into segments
- store data in database
- use simple classifier, which can authenticate the user
- visualize raw data
- prepare a demo for product presentation

### 2.3. Our goals in the end of summer semester:

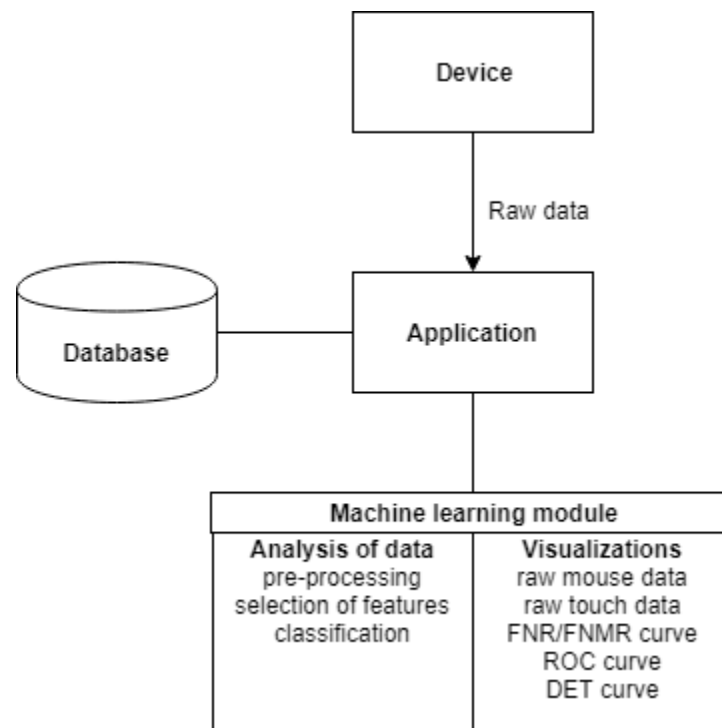
- visualize data from mouse and mobile
  - highlight segments
    - in touch and also gyroscope and accelerometer data
  - include custom columns
  - add background image to diagram
  - discrete or continuous display of data
- create pipeline
  - extract movement data
  - split into segments
  - create features
  - clasification
- demo for product presentation

## 2.4. Review of goals for winter and summer semester

Our project provides a functional logger for events from a computer mouse and also for events from a mobile device. This data is stored in a database on the server. Our product also provides data pre-processing and performs a simple classification. Machine learning module provides visualization functions for displaying data from a computer mouse and also data from a mobile device. We created pipeline and a demo for our application to present our product.

## 2.5. The view of components connection in the system

Raw data in our program are events for which parameters are recorded - e.g. for a computer mouse, they are x and y coordinates of the mouse on the screen, time stamp, event type.



*Fig. 1 : The view of components in the system*

### Device

Personal user device, e.g. smartphone, notebook from which we collect data about user activity.

### Application

Application module which stores raw data from devices in database, provides data for further analysis and visualisation.

### **Database**

Storage of raw data from devices.

### **Machine learning module**

Module for raw data pre-processing, feature computation, user authentication and data visualisations.

### **Web interface**

Web app for visualisation of raw data or results of data analysis.

# 3. Modules of the system

## 3.1. Server module

The server module provides an interface for storing and querying data from database. After accepting the data storage requirement, the server extracts metadata and then stores these data together with the metadata in the database. It also provides the interface for obtaining information about the data that is currently stored in the database (for now, it is only the names of columns). It can export *.csv* file, which contains all stored data. The server adjusts the response of the demand when querying and exporting. This means that we can retrieve only a subset of data stored in the database (certain stamps, certain values).

Server API documentation is available at:

<https://tp-fastar.gitlab.io/server/>

## 3.2. Logger module

FastarLogger is a JavaScript library used to record mouse events and events from device sensors (accelerometer, gyroscope). It allows to log pressing and releasing mouse buttons (left, middle, right, forward and back), mouse movements, angular velocity and acceleration of the device along all three axes.

Functionality of logger is oriented in following files:

- `logger.js` - class which contains logging logic and corresponding methods
- `connection.service.js` - file contains functions used to communication with backend server
- `index.spec.js` - file contains all logger unit tests

Module use *WebPack* to compile *JavaScript* modules and build library to *UMD* format. To transpile *ES6* code to *ES5* for better compatibility *Babel* is used. We use frameworks *Mocha* and *Chai* to write unit tests.

Example of logged data:

```
{
  "eventType": "MOUSE_MOVE",
  "payload": {
    "mouseButton": "LEFT_BUTTON",
    "positionX": 305,
    "positionY": 582
  },
  "sessionId": "5bf43285e6bdb1000b645890",
  "sessionStartsAt": 1542730372913,
```

```
"timeStamp": 13372.699999999895  
}
```

<b>Property</b>	<b>Description</b>
eventType	type of logged event
payload	additional event data (depends on eventType)
sessionId	unique session identifier (obtained from BE on logger initialization)
sessionStartsAt	time of session start in milliseconds
timeStamp	time of event invocation in mili-seconds

Each logger initialization represents new session with unique session id obtained from backend server.

Logged data are sent to backend server in batches of 100 logs (default).

### **3.3. Machine learning module**

Machine learning module is responsible for data preprocessing, feature extraction and user authentication. It is written in language Python3, using libraries sklearn, pandas, numpy and bokeh.

Machine learning module provides functionality to visualize data from computer mouse and mobile device. In these function you have multiple options - to highlight segments (highlighting in touch, gyroscope and accelerometer data simultaneously), choose between discrete or continuous display of data, show customs column in figure and add background image to displayed data. Using the module you can create a pipeline, which extracts data from movement, split them into segments, create features from them and then classifies into classes.

This module contains the following packages:

- authentication
- estimator
- features
- preprocessing
- utils
- visualization



# 4. Manuals

## 4.1. Manual for product deployment

Developed product consists of the following components:

- [Server](#)
- [Logger-web](#)

These parts are developed in various technologies, so different procedures are required when using / deploying them.

### 4.1.1. Server

Docker is used to deploy the app. It is needed to create a docker image locally or download a docker image created after the end of the docker registry from the gitlab.

**For the following instructions, you need to have a docker installed!**

#### Local deployment

##### Local image creation

1. You have to be in the project folder
2. Image creation

```
docker build -t registry.gitlab.com/tp-fastar/server:<version> .
```

##### Download the image from the registry

1. Login

```
docker login registry.gitlab.com (login credentials as on gitlab)
```

2. Download

```
docker pull registry.gitlab.com/tp-fastar/server:<version-or-latest>
```

##### Launching

1. `docker run --name behametrics-server -d -p <local-port>:<container-port> registry.gitlab.com/tp-fastar/server:<version-or-latest> - container-port is the port number on which the application is running in the containers. This number is always in Dockerfile - EXPOSE`
2. Application is running locally on port **local-port**

## Stop

1. `docker stop behametrics-server`

## Required configuration

All configuration is done using environment variables. Currently used environment variables:

- `DEBUG` - Flask debug configuration; defaults to `True`
- `PORT` - HTTP port; defaults to `5000`
- `MONGO_URI` - URI used for connecting to mongo instance; defaults to <mongodb://localhost/behametrics>

## Easy deployment for end user

Two *Docker Compose* files are provided for convenience.


1. Configuration which expects `MONGO_URI` for database connection and exposes endpoints on port 5000
2. Configuration which is using internal Mongo container (service) and exposes endpoints on port 5000

Using this configuration you are able to run everything in one command but be cautious of your data. First configuration is preferred as you are required to provide own Mongo database.

## Deployment on server

The server is deployed using CI / CD processes that are defined in `.gitlab-ci.yml` and have stage: `deploy`. Each of these processes has a defined environment to which it is deployed. An overview of the environments along with the current deployment is available in the Operations → Environments section of the gitlab repository. In exceptional cases, the app may be deployed to the server manually.

In the following picture is an example of an environment called "**production**" along with a list of processes that have been deployed to that environment. The process always deploys the version according to **the number of pipeline** in which it runs. Version deployed by individual processes we can find out by clicking on the number of processes - for example # 109206898.

Re-deployment of the current / previous version can be done using the button  on the right of the screen.

ID	Commit	Job	Created
#43	Y [commit hash]	Full deploy (#109206898) by [user]	2 days ago
#42	Y [commit hash]	Full deploy (#108925722) by [user]	3 days ago
#41	Y [commit hash]	Full deploy (#108924092) by [user]	3 days ago

*Fig. 2: Example of deployment of solution*

## 4.1.2. Logger-web

Web logger is a library written in JavaScript. The library itself does not need to be deployed as a production application to the server. It is just needed to provide library files in git repositories. It is necessary to provide available functional library.

In order to make the executable and functional version available, we use CI / CD processes.

The definitions and sequences of these processes are written in **.gitlab-ci.yml**.

After successfully passing the processes, the artifacts are available in the form of the library itself, in a folder **.lib**.

### Scripts

`npm run build` - produces production version of the library under the lib folder

`npm run dev` - produces development version of the library and runs a watcher

`npm run test` - runs the tests

`npm run test:watch` - runs the tests in a watch mode

`npm run demo` - runs demo page to show logger in action

### Logger configuration

Developer is able to configure logging configuration. When instance of Logger class is created, we can pass configuration object to class constructor (see example below).

```
let logger = new Logger({
  apiUrl: 'https://your-domain.com'
})
```

**Configuration values:**

<b>Name</b>	<b>Type</b>	<b>Description</b>	<b>Defaults</b>
apiUrl	string	Base API URL to post logged data and get session id	
mouseEvents	array<string>	List of mouse events to log	['mousemove', 'mousedown', 'mouseup', 'wheel']
touchEvents	array<string>	List of touch events to log	['touchstart', 'touchend', 'touchmove', 'touchcancel']
sensors	array<string>	List of sensors to log	['gyroscope', 'accelerometer']
batchSize	number	Number of logs sent in one POST request	100
logToConsole	boolean	If true all logs are shown in browser dev console	false
gyroscopeConfig	Object	Configuration object for gyroscope sensor	{frequency: 60}
accelerometerConfig	Object	Configuration object for accelerometer sensor	{frequency: 60}

### **Payload values:**

Mouse events payload:

*mouseButton* - describes which button is pressed during events

*positionX* - provides the horizontal coordinate (offset) of the mouse pointer in global (screen) coordinates

*positionY* - provides the vertical coordinate (offset) of the mouse pointer in global (screen) coordinates

### **Wheel event:**

*mouseButton* - describes which button is pressed during events

*positionX* - provides the horizontal coordinate (offset) of the mouse pointer in global (screen) coordinates

*positionY* - provides the vertical coordinate (offset) of the mouse pointer in global (screen) coordinates

*scrollDeltaX* - double representing the horizontal scroll amount

*scrollDeltaY* - double representing the vertical scroll amount

### **Touch events payload:**

*touches* - represents a list of contact points on a touch surface

*force* - amount of pressure being applied to the surface by the user, as a float between 0.0 (no pressure) and 1.0 (maximum pressure)

*id* - unique identifier of touch object in touches list

*positionX* - horizontal coordinate of the touch point relative to the left edge of the screen

*positionY* - vertical coordinate of the touch point relative to the left edge of the screen

### **Gyroscope payload:**

*alpha* - double containing the angular velocity of the device along the device's x axis

*beta* - double containing the angular velocity of the device along the device's y axis,

*gamma* - double containing the angular velocity of the device along the device's z axis

### **Accelerometer payload:**

*accX* - double containing the acceleration of the device along the device's x axis

*accY* - double containing the acceleration of the device along the device's y axis

*accZ* - double containing the acceleration of the device along the device's z axis

## 5. User testing

We have prepared demo jupyter notebooks with pipeline and visualizations of mobile and mouse data. User had simple task to add highlighting of segments in the showed figure. The second step for the user was to fill in the questionnaire.

We have tested 2 demo jupyter notebooks with visualizations (mouse and mobile data) on 3 users. As a first step, we explained to our testers what we do and what features our products provides. After brief look on the jupyter notebook with mouse data, we gave him task to add highlighting of segments. At first, all of them had problem to find a function to highlight, because they searched for keyword segments instead of highlight. We gave our testers a little hint to search for another keyword and second try for all of them was successfull. Also in the second notebook with mobile data, they had no problem to finish the same task.

From given questionnaires we found out that all testers would appreciate the library behavioral data visualization and also they would use our functions in some of their projects. We got one suggestion to show segments on default, not just on hover of the mouse.

Another questionnaire was about FastarLogger and FastarServer. We had one respondent, he answered that installation of logger module was easy, even though we had not read README file to module. We asked him if FastarLogger library would facilitate behavioral biometrics work, he aswered yes, because it was easy to use and recorded the necessary data. The part about FastarServer was not so positive. He claimed that the installation was not so easy and also that we had little experience with the module. He also said, that he was initially unable to access the logged data because of poor documentation.

Questionnaire for logger and server modules:

- <https://forms.gle/bUbe3H9UeJqfdMJg6>

Questionnaire for visualizations of mobile and mouse data:

- <https://forms.gle/q4RKGzCUiCngEFiZ9>

## 6. Technical documentation

Server module: <https://tp-fastar.gitlab.io/server/>

Logger module: <https://tp-fastar.gitlab.io/logger-web/doc/>

Machine learning module: <https://tp-fastar.gitlab.io/ML-module/>