Using MEAN stack for development of GUI in real-time big data architecture

Marko Štajcer, Marko Štajcer i Dražen Oreščanin
Poslovna Inteligencija d.o.o., Zagreb, Croatia
{marko.stajcer, stajcer.marko, drazen.orescanin}@inteligencija.com

Abstract – There are several key criteria for successful software development branch and one of them is architecture. We describe in detail architecture, design and development of graphic user interface for real-time fraud detection system in the telecom industry. Our focus is on a web application architecture, which includes data model component, technical infrastructure component, and components that interact or are associated in any way with the system (users, third party components).

I. INTRODUCTION

Complex event processing, or CEP, is event processing that combines data from multiple sources [1] to infer events or patterns that suggest more complicated circumstances. The goal of complex event processing is to identify meaningful events (such as opportunities or threats) [2] and respond to them as quickly as possible. Many of today’s big data technologies were built on the idea of batch processing human generated unstructured and predominantly persistent data, stored in distributed file systems. However, data volumes generated from machine-to-machine interactions (M2M) surpass by far the amount of data generated by humans. M2M data streams must be processed in real-time or near real-time, they are predominantly transient and naturally distributed, and are much more structured in nature. FERARI is a collaborative project within the European Commission’s FP7 ICT Work Programme, whose goal is to pave the way for efficient, real-time big data technologies of the future. It will enable business users to express complex analytics tasks through a high-level declarative language that supports distributed complex event processing and sophisticated machine learning operators as an integral part of the system architecture. Effective, real-time execution at scale will be achieved by making the sensor layer a first-class citizen in distributed streaming architectures and leveraging in-situ data processing as a first (and, in the long run, the only realistic) choice for realizing planetary-scale big data systems.

To give a new potential user a good idea of the capabilities provided by the FERARI platform, we created an example application. This application demonstrates a basic mobile fraud detection scenario, modeled after our mobile fraud use case scenario. Nevertheless, the prototype is built upon the FERARI open source architecture and strongly interacts with it. We start with the system architecture that we are using for the application development. In the next section we continue with technology that we are using including their advantages in modern web development and with the description of the graphic user interface.

II. ARCHITECTURAL INTEGRATION OF FRAUD MINING PROTOTYPE

A. System architecture

Our focus is on the web application architecture, which includes the following:

- data model component
- technical infrastructure component
- components that interact or are associated in any way with the system (users, third party components)

To build a stable ecosystem for web application, we have to consider the needs of the functional graphical user interface:

- collecting data from data source
- handling data
- sending data to the front-end
- getting data and displaying on the front-end

In order to achieve the previously mentioned, we need a solution, or more, to develop an application with a structure broken down to database, server and client. There are several benefits we can achieve through breaking down the whole application into smaller and modular components:

- easier application maintenance and expansion
- reducing the number of bugs
- reusable components (using individual components in many different applications), parallel programming
- minimal impact when replacing the technology
- security significance. [3]

Because of that, the application is built in three-layer architecture, as shown in Figure 1.
**B. Presentation layer**

The presentation layer gives a user the possibility to interact with the application through elements such as navigation, buttons, pop-up windows and other elements for managing user interaction which have been implemented by user interface components and presentation logic.

User interface elements/components actually include information which we want to show in a particular form regarding specification of the user interface for the application. Some of these elements are used to display information to the user and to accept user input. In some architectural patterns there are UI components called views, whose role is to show the presentation to the user, regarding underlying data and the application logic. The user interface components can be extended with logic which is common in the user interface implementation. The extension of the UI components should not be exaggerated with previously mentioned logic.

The application code which cares about the behavior and structure of the application (user interface code) is placed in the presentation logic. The behavior can be defined through units called user stories, and we have logic components for their implementation. There are two more roles of the logic components:

- controlling the user’s interactions with the underlying logic and state of the application
- formatting data from the business layer for the UI components.

**C. Application business logic layer**

The core application logic and the bridge between the presentation and the data layer represent the business logic layer. When we say bridge, it means that it gets data from the data layer and sends the results to the presentation layer. Within the context of the FERARI real-time processing, the actual business logic is expressed in terms of CEP expressions that are evaluated in real-time, in a distributed manner. In the considered use case, the FERARI processing system is generating alerts that relate to the detection of possibly fraudulent events. This “business logic” is of course outside of the GUI application. However, the end-users requested to have the possibility to store these alarms, work on the history of alarms, and investigate fraudulent events in an interactive manner, after the corresponding alarm has been raised. For this purpose, we added a data access layer for storing alarms and short term history of events. The business logic for working on stored alarms and the short term history is implemented in the GUI application.

**D. Data access layer**

The data access layer (the DAL) is responsible for tasks related to reading and writing data, which uses technologies and programs (e.g., SQL, JSON parser, ODBC, etc.) for the data access logic implementation. There is, the presentation layer which defines the user interface, and the business layer in which the behavior of the application is defined, and at the end there is the data access layer for data flow control. Each of these layers communicates only with layer above or below it and because of that, there is no connection between data and presentation layer.

**E. MEAN stack**

MEAN is a free and open source powerful JavaScript framework which simplifies and accelerates full stack web application development. MEAN stands for:

- **MongoDB** – very popular NoSQL database solution designed with web application in mind the user’s interactions with the underlying logic and state of the application
- **Express** – web server framework for Node.js
- **AngularJS** – frontend JavaScript MVC framework for single-page application development
- **Node.js** – JavaScript server platform built on V8 JavaScript engine

The most interesting fact is that the MEAN Stack is using only one programming language (JavaScript) on each layer of the application. We looked for a modular solution that will give us flexibility needed for the project. We have chosen MEAN, but we are not using original MEAN Stack with the mentioned technology. We have replaced MongoDB with MySQL and there is a MEAN (*MySQL, Express, AngularJS, Node.js) stack. [4]

**F. Single page applications — modern web applications**

In web applications development, there are different approaches based on business logic placement:

- multi-page application
- single-page application (SPA).
Multi-page application: Back-end side contains most of the business logic with a little bit of development on the client-side (JavaScript). The example where the most of the business logic is implemented on back-end side is the Ruby on Rails applications (MVC architecture). In this case, when user changes a state through the web page navigation, every change sends a HTTP request.

Single-page application (SPA): Client side contains most of the business logic which is often developed by front-end MVC architecture. In this case, there is a front-end router whose role is to deliver AJAX requests and make API calls to the back end.[6]

SPA is an application that keeps the web page from reloading during use. The purpose of the application is to help the user to complete a task, for example; write a document, or administer a web server. SPA can be considered a thick client loaded from the web server. SPA delivers a desktop application in the browser; which results in a highly responsive experience, and a bigger number of satisfied users. There is a difference between SPA clients and traditional web pages. The replacement of a traditional website with a SPA causes the change of the entire software; from the database server to the HTML templating. The companies that aimed at a successful transition from the traditional website to the SPA, have understood that the change of the old structures and practices is necessary, so they decided to refocus the discipline, the engineering and the testing on the client. The server does not lose its importance, but focuses on providing JSON data services. Here are some important advantages of JavaScript SPAs over Java and Flash:

- No plugin required – There is no need for plugin when users want to access the application, and there is also no need for the developers to worry about a separate security model, which facilitates the use.

- Less bloat – An SPA using JavaScript and HTML usually uses fewer resources and does not require an additional run-time environment, which is necessary when it comes to plugin.

- One client language – Web architects and most developers are obliged to be familiar with many languages and data formats – HTML, CSS, JSON, XML, JavaScript, SQL, PHP, Java, Ruby, Perl, and so on. There is no need for writing applets in Java, or Flash applications in ActionScript, when JavaScript is already in use elsewhere on our pages. By using a single programming language for everything on the client, we significantly reduce complexity.

- A more fluid and interactive page – A Flash or Java application has already been seen many times on a web page. There is often a big difference in details between the HTML elements that surround it, and the application itself, which is usually displayed somewhere in the box. The said implies that some elements like the graphical widgets, the right click, and the sounds are very much different, while the interaction with the rest of the page is limited. With a JavaScript SPA, the entire browser window is the application interface.[11]

With the maturation of JavaScript[11], most of its weaknesses have been either fixed or mitigated and, causing the increase in value of its advantages:

- The web browser is the world’s most widely used application – The majority of people leave the browser window open throughout the day, to make sure they can use it whenever they want. The access to a JavaScript application is only a bookmark click away.

- JavaScript in the browser is one of the world’s most widely distributed execution environments. In the last 3 years, the increase in the activation of android and iOS mobile phones caused a significant increase in shipping of robust JavaScript implementations on phones, tablets, laptops, and desktop computers around the world.

- JavaScript deployment is trivial – By hosting it on a HTTP server, we could make the JavaScript application available for more than a billion web users.

- JavaScript is useful for cross-platform development – Now SPAs can be created using Windows, Mac OS X, or Linux, and a single application can be deployed not only to all desktop machines but also to tablets and smart phones. Mature libraries such as jQuery and PhoneGap that smooth over inconsistencies and the convergence of the implementations of standards across browsers have contributed to the mentioned.

- JavaScript has become surprisingly fast and can, at times, rival compiled languages – For its speed up, we can thank to an ongoing and heated competition between Mozilla Firefox, Google Chrome, Opera, and Microsoft. Modern JavaScript implementations have experienced advanced optimizations such as JIT compilation to native machine code, branch prediction, type-inference, and multi-threading.

- The evolution of JavaScript has enabled it to include advanced features – These features include the JSON native object, native jQuery-style selectors, and more consistent AJAX capabilities. Mature libraries like Strophe and Socket.IO have made push messaging far easier.

- There has been a significant advance in HTML5, SVG, and CSS3 standards and support – thanks to these advancements, the rendering of pixel-perfect graphics that can rival the speed and quality produced by Java or Flash, is allowed.

- JavaScript can be used throughout a web project – Now the excellent Node.js web server, and data stores such as CouchDB or MongoDB can be used, both of which communicate in JSON. Even libraries can be shared between the server and the browser.
The power of the desktop, laptop, and even mobile devices has increased. The omnipresence of multi-core processors and gigabytes of RAM allows the processing that used to be accomplished on the server to be distributed to the client browsers.

The popularity of JavaScript SPAs has increased due to these advantages, and the demand for experienced JavaScript developers and architects has also increased. Applications that were previously developed for many operating systems (or for Java or Flash) are nowadays delivered as a single JavaScript SPA. Startups have chosen Node.js as the web server of choice, while JavaScript and PhoneGap are used by mobile applications developers to create “native” applications for multiple mobile platforms using a single code base. [11]

G. MVC design pattern
For better organization of the web application architecture there is a widely adopted structure Model View Controller (MVC), popular architectural pattern whose idea is to separate code into three components of the application.

- **Model** stores data retrieved from the controller and displayed in the view
- **View** represents a presentation based on the data in the model
- **Controller** contains business logic and defines the behavior of the application

The controller is activated when end user interacts with the application. At this moment, the data in the model are changed by controller and the view can get these data to display them to the end user. The view can make an update when it knows that a change has occurred by notification from the model. [9]

AngularJS is an open source JavaScript library, whose development and maintenance is Google’s responsibility. Some of the largest and most complex web apps around have been using it. The MVC pattern is followed by the AngularJS applications, while the development process itself relies on a wider range of building blocks. There are the headline building blocks - the model, the views, and the controllers - but there are also numerous moving parts in an AngularJS app, as well as modules, directives, filters, factories, and services. [8]

H. Application flow
We are going more in detail with the architecture of the application and we will explain the connection among technologies used in separate layers. With separation of our application we achieve the possibility that each layer can handle specific tasks. On the back-end side there is a server (node) whose role is to handle data layer and provide information to the front-end on the client side (angular). As shown in Figure 3, on the front-end we have angular which makes a request (API call) to the server. Our server stands alone with API from which angular can get all data. In order to access a database we need to have the application connected to it. When the node has access, we can build routes and define API and at the end JSON information is provided to the angular. The whole concept (shown in Figure 3) allows us to separate the front-end part of the application from the data (API).

I. Realtime architecture
If we want to show some data to the front-end in real-time we need a solution which includes pushing the same data from the back-end to the front-end side when something new (fraud event in our case) happens. In this case, the most of the logic is on the front-end side, because the client has to retrieve and load data dynamically, and change the user interface. Figure 4 depicts the overview on the real time push architecture we made, based on the FERARI open source architecture.
Inside of the red container the story begins with external data stream source which emits data tuples to the DetailOut bolt. DetailOut bolt's role is to cache a limited amount of detailed data in RDBMS (short term history). This information is needed for the purpose of forensic data analysis and validation of this specific use case. We want to store data only for this specific cases to make validation of detected fraud events and previously mentioned action will provide better insight into events. Apart from those details only derived events are stored until they are processed by the fraud mining expert. This caching mechanism for details is necessary as it is impossible for the streaming real time implementation to store all call details. Nevertheless, the capability to process all incoming stream items is not influenced by this reduction.

Next we have the components of the FERARI architecture: CEP (complex event processing) engine that is helping with the development of complex event processing applications. It is used for real-time event data analyzing and generating immediate insight. CEP (including its technology and implementations) also includes enabling instant response to changing conditions and providing resources to define and maintain the event processing logic of the application. Flexibility of using CEP comes from its runtime that can meet functional and non-functional requirements without affecting application performance and there is also no need for the developers and system managers to worry about the issue, which facilitates the use. Complex event processing component is built on and extends the IBM Proactive Technology Online (PROTON) in the FERARI project. The CEP is responsible for filtering and combining data from external sources and at the end for serving meaningful results. These results are processed in the FraudEvent which publishes data to a dedicated channel in Redis, as the event occurs. As being subscribed to the same Redis channel, the Node.js receives all the events as they occur, as well as all the other subscribers subscribed to the Redis channel. In order to enable communication and data streams between the server and the browser we have to implement something that provides real-time updates. For Node.js there is a Socket.IO module which has communication methods both on the server and the client, and as the Figure 4 shows, we are using these methods to enable communication on both sides. Node.js is subscribed to Redis channel and it listens to the message 1. When the message comes, Socket.IO allows the sending of the message back to the browser clients that are connected to the server and subscribed to the same channel. Browser will display the content of the message when FraudEvent bolt publishes the message using the communicator. In the current state of the prototype we use Redis to provide the actual communication channel. The benefits of that mechanism are recognized in connecting and providing subscribe from every third party system without any additional knowledge or change in the rest of the technology used in the architecture in order to receive a message. It is enough to have access to the Redis channel and subscribe to the same. If compared to a traditional messaging system this can be a disadvantage.
but it is advantageous where data has importance in real time and need not to be stored.

Redis is a NoSQL fast in-memory key-value store which is used when performance and scaling are important. It is currently used in the prototype as underlying communication layer. Access to this layer is modularized via the FERARI interfaces such that it can be replaced in a later stage with the advanced communication layer. For the purpose of caching recent fraud events and the limited amount of corresponding detail records, the in-memory storage system of Redis is well suited. Besides the mentioned, it can do many different things, for example: implementing a pub/sub system. As the keys can contain strings, lists, sets, hashes and other data structures, Redis is also known as a data structure server.

Pub/sub stands for publish-subscribe, a pattern where messages are not sent directly to specific receivers. The publishers send messages to different channels, and the subscribers receive these messages only if they are listening to a given channel. The pub/sub pattern is supported by Redis, responsible for the commands that allow the publishing of messages and the subscription to the channels. [7]

Ever since there has been an onset of web applications, developers have worked on different ways of getting duplex communication between the server and the browser. The aim of Java, Flash, Comet, and other workarounds, is the same. But for the first time, there is a specification to build a full-duplex communication system, by using HTML5 WebSockets. HTML5 specification contains an astonishing new communication system, a feature which can define a full-duplex communication channel operating over the Web through a single socket. Although the WebSockets RFC is published, the older browsers that are still in use are not, and will never be able to obtain it. Socket.io is an abstraction layer for WebSockets, which has Flash, XHR, JSONP, and HTML File fallbacks. An easy server and client library, which is used for making real-time, streaming updates between a web server and a browser client, is also provided by the Socket.IO. [12]

### III. CONCLUSION

This article presents a big prototype platform for fraud detection, and gives an overview of the architectural integration in combination with multiple technologies, frameworks and techniques through three-layer architecture. It is shown how the M*EAN stack solution supports development of a graphical user interface with quantitative and other types of charts that give some general information about the fraudulent. Since fraud continues to be a very real problem in the telecom industry, presented big data architecture has great potential in identification of suspicious calls and users.

### REFERENCES

[1] Schmerken, Ivy (May 15, 2008), Deciphering the Myths Around Complex Event Processing, Wall Street & Technology


