Kirill Kazantsev

Development of a Web Application for Management of Public Events

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

April 15, 2018

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Kirill Kazantsev Development of a Web Application for Management of Public Events 49 pages 15 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kimmo Sauren, Senior Lecturer

This thesis was carried out to study the topic of Full Stack JavaScript development and based on the research to implement a web application for management of public events using the studies technologies.

A thorough analysis of the topic was conducted. Such technologies, as Node.js, MongoDB and Express were examined in great detail. Their advantages and disadvantages were identified and compared to other technologies that are used in web development. After a prolonged research, a working web application prototype for management of dance tournaments was developed. The prototype consists of the creation of tailored tournaments using an appealing user-friendly interface, a possibility to easily sign up for tournaments and make payments and, finally, a management of all tournaments and registered participants

The application was built based on the studied technologies as well as a custom framework that was created on top of standard “express” router to have a better control and flexibility. The project suggested a way of organizing the directories and designing the architecture.

The thesis concludes with the thoughts about the technologies used in the development and different sides that could be improved as well as different frameworks and approaches that could be utilized.

Keywords	Node.js, MongoDB, Express, JavaScript

Contents

Abbreviations

AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheets
DBMS	Database Management System
HTML	HyperText Markup language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
IDE	Integrated Development Environment
IIS	Internet Information Services
RDBMS	Relational Database Management Systems
SPA	Single Page Application
SQL	Structured Query Language
URL	Uniform Resource Locator

1 Introduction

Only a few decades ago, the society was overwhelmed by the idea of an era of advanced technologies. Today it is no longer an idea, it is a reality. Nowadays, the Internet is the most rapidly developing environment of information exchange in the history of mankind. It is difficult to name any area of activity that would not have its full and comprehensive reflection on the Internet.

However, that all would be hard to imagine without web applications. In 1995 Netscape presented a client-side scripting language called JavaScript and starting from that moment the history of web applications can be traced. Nevertheless, the concept “web application” itself was introduced only in 1999. Generally, web applications have evolved from static web pages to dynamic websites that provide tailored user experience based on information known or given by user. Web applications are used for a big variety of purposes, for example, appointment systems, instant messaging services, commerce, entertainment and even democratic decision making in some countries.

This thesis focuses on the implementation of the web application for management of dance tournaments. The application allows users to create tailored tournaments using an appealing user-friendly interface on web browser. At the same time, the application gives a possibility to easily sign up for tournaments and make payments. It can also be used to manage all the tournaments and registered participants. The data and functionality that is available for users is based on different access rights. All the data related to the application is stored in a database, that is located in the server-side. The server receives and processes all the data that is sent from the web browser and sends back all the requested data.

2 Theoretical background

2.1 Web application

Web application is a client-server application where a browser acts as the client and a web-server as the server. Web application logic is distributed between the client and the server, data storage is performed mainly on the server. Data is exchanged over the network using Hypertext Transfer Protocol (HTTP). One of the advantages of this approach is the fact that users do not depend on a specific operating system or hardware configuration. Hence, web applications are cross-platform services.

2.2 Front-end or client-side

Front-end development is a process of creation of a public part of a website, which is in direct contact with the user. Someone can say, that front-end development is only about making the website pretty and appealing [1]. It is definitely a very important part of the process, but there are a lot of different technologies that fall within the scope of client-side development [1]. The core tools that are used in the process are: HyperText Markup language (HTML), Cascading Style Sheets (CSS) and JavaScript (JS).

HTML is the foundation of a webpage. It is a markup language, which defines the overall structure of the page. [1] Using element or tags such as list, table, header, the developer can label different parts of the content [2]. HTML language is interpreted by browsers and then the formatted text resulting from the interpretation is displayed on a computer screen or mobile device.

CSS allows web designers and developers to style and tune every component that is defined in HTML [3, 455]. CSS was primarily intended to enforce the separation of webpage content from webpage style, including features such as fonts, colors and layout. This separation enables several HTML documents to share the style specified in a separate .css file. [3]

JavaScript is the backbone for every dynamic and interactive functionality on the webpage. It is a lightweight and interpreted programming language [4]. It can be used, for example, to check the validity of the data that the user has entered or change the

structure of a webpage based on the trapped user-initiated events, such as mouse clicks. Moreover, one of the most useful features in JS is Asynchronous JavaScript and XML (AJAX), that technique can be used to send to and receive from a server all the required data without refreshing the webpage.

2.3 Back-end

Back-end development refers to the implementation of server-side, which primarily focuses on web application logic or, in other words, how the application works. It is a process of creating the core of web application, developing the platform for the application and filling it with all the required functionality. Server-side manages the data that is received from the front-end and returns the result back in the form that is understandable by the client-side. Back-end usually consists of three main parts: a web server software, an application logic and a database.

2.3.1 Web server software

Web server software is a program that runs on the hardware and serves data for the clients that are usually represented as browsers. Web server software consists of several parts, but the core is an HTTP server [5]. It is a software that knows what is a Uniform Resource Locator (URL) and understands the HTTP protocol [5]. The basic principle of client-server communication using HTTP can be demonstrated by the figure below.

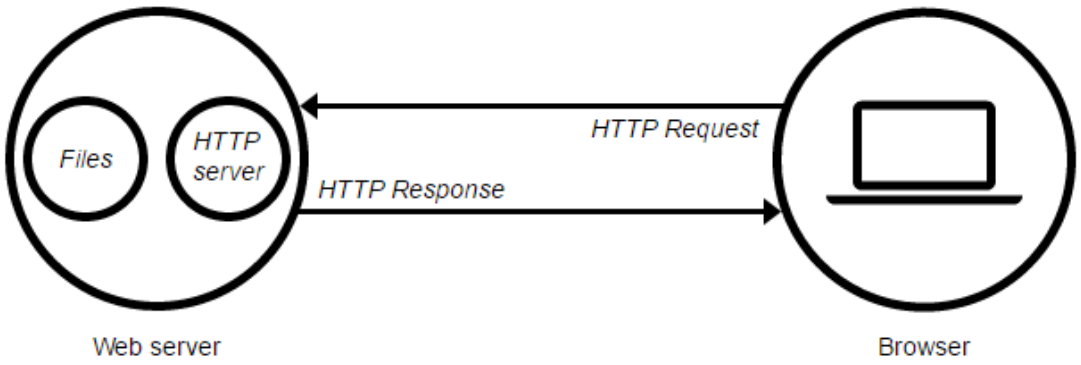

Figure 1. Client-server HTTP communication
Copied from MDN [5]

As Figure 1 shows, the browser sends the request, then it reaches the dedicated hardware and then HTTP server sends a response containing the information that was requested. When HTTP protocol is used certain set of rules is always followed [5]. First of all, server is able to only respond to the request that was sent by the client. It cannot send requests to the browser. Secondly, the server must send a response to every incoming request, at least a response that contains an error message. [5] Overwise, the client will receive a timeout message, that indicates that the server has not answered within the specified time. Finally, every HTTP request must contain a URL address that indicates the particular server and path that this request should be sent to [5].

There are many different options to choose from when selecting a web server. Among them are: Internet Information Services (IIS) from Microsoft, nginx from NGINX, Apache HTTP Server, Google Web Server and etc. The market share of every web server as of September 2016, can be seen in the following figure.

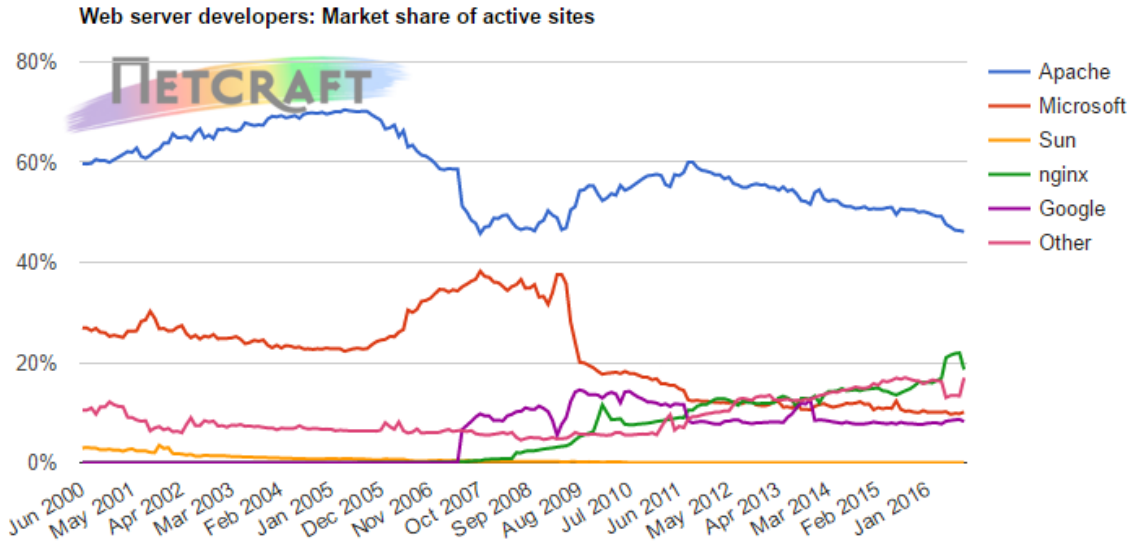

Figure 2. Web server developers: Market share of active sites
Copied from Netcraft [6]

As Figure 2 demonstrates, the most popular web server at the moment is Apache with more than 46% of the market share. The next popular server is nginx with around 19%. The selection of a web server is based on different aspects, such as the operating system, the level of security and performance power.

2.3.2 Application logic

Application logic is also called business logic of the server. It involves all the operations on processing the requested and sent data, saving the data to the database, making decisions on what data is required and then querying the necessary data from the database. It allows to utilize security measures, such as, creating authentication mechanisms to identify the user that is requesting or sending data from and to the server. Authorization is another example of the security measure to be implemented in application logic. It is used to verify if this particular user has the necessary rights to perform the desired operation.

Business logic is created programmatically using different programming languages such as Java, PHP, Ruby, Python, .Net and JS in case with the use of Node.js server software. Nowadays it is almost always done with the help of server-side frameworks. Web application frameworks ease the development by covering and providing some of the core functionality that is commonly performed such as session management, authentication mechanisms, output formatting, database interaction and etc [8]. The example of a server-side framework can be CakePHP, Ruby on Rails, Express, Spring, Django and etc.

Thereby, based on the foresaid, application logic is one of the most crucial parts of back-end development that prescribes the ways how the data can be displayed, created, saved and manipulated. The undeniable importance and the overall position of the business logic in the application development can be demonstrated by the figure below.

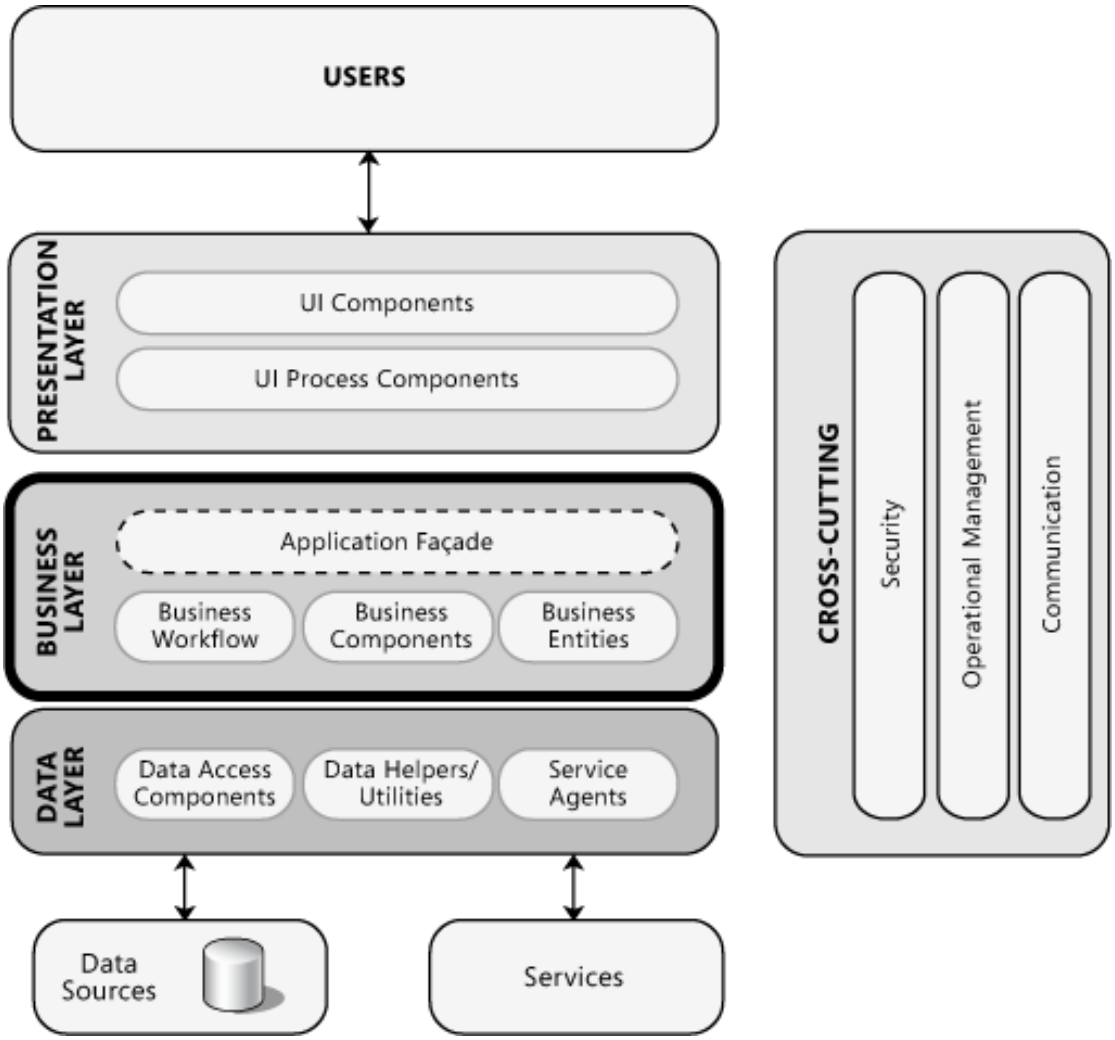

Figure 3. Typical application structure
Copied from MSDN [7]

Based on Figure 3, it can be seen that business logic takes one of the most vital parts in the application structure. Implementation of all the needed rules, limitations and process sequences is accomplished by encoding the real-world business rules into the code understandable by machines. Developers create special workflows as a part of the application logic to determine how different business objects should interact with each other to achieve the necessary result. [7]

2.3.3 Database

Database Management System (DBMS) plays an extremely important role in the web application development in general and as a part of the back-end in particular. DBMS

gives a possibility for saving, modification and deletion of the data. It allows to query (retrieve) data from the actual database for further processing by the application logic layer. DBMS provides ways to define how the data is organized, enforce different security measures, maintain data integrity, create and control concurrency rules, add and monitor users and etc. There are two fundamentally different types of databases: relational and non-relational.

Relational Database Management Systems (RDBMS) have been absolutely the most popular database to utilize for over 20 years [9]. All relation databases use Structured Query Language (SQL) as the language for manipulating data and administrating the database. Therefore, relational databases are very often called SQL databases. SQL databases store data in tables [9]. Inside the tables the data is represented in the form of rows and columns where rows are entries and columns are attributes [9]. The data is highly structured and storage schema has to be very strict [9]. Different tables maintain certain kind of logical connections, called relationships. This kind of relations is created on the basis of specified interaction between these tables. Thus, this is where the term “relational database” came from. Relational databases are usually used in the case where flexibility concedes the strict scheme and the vertical approach with more resources for one server is mainly used to achieve scaling [9]. Some of the most popular RDBMS are: Oracle, MySQL, SQL Server, PostgreSQL and etc.

Non-relational database popularity started to grow quite recently. According to Leavitt N. (2010), one of the most significant raises in popularity for non-relational databases happened in 2007, when Amazon published a work describing the distributed non-relational databases system called Dynamo which was created for internal use [10]. Non-relational databases do not usually use SQL as the language to for data management. Therefore, they are very often referred to as NoSQL (Not Only SQL) databases. NoSQL type of the databases is profoundly different from ordinary Relational Database Management Systems [9]. Comparing the tabular data storage used by the RDBMS, NoSQL databases stores data in various different formats such as key/value pairs, graphs, wide columns, JSON or document format and etc. The data can be very dynamic and storage schemas are completely flexible. As a benefit of such flexibility every record can have different properties and new properties can be easily added whenever necessary without the need to alter the schema used by the database, because the schema is dictated not by the database, but by the application [9]. Therefore, NoSQL databases are good in the cases

where the data is complex, nested or not highly structed and the scaling is usually obtained by partitioning data to span servers meaning horizontal approach is taken [9]. MongoDB, Apache Cassandra, Redis, Apache HBase and neo4j are among the most popular NoSQL databases at the moment.

3 Technologies

3.1 Node.js

Node.js is an open-source MIT licensed JavaScript runtime environment designed for creating a huge variety of scalable server applications. It is aimed for asynchronous and event driven programming model that makes it efficient and lightweight. [11] Node.js was created by Ryan Dahl in 2009.

3.1.1 Architecture

Node.js is built on top of V8 JavaScript engine. The same one that is used in Google Chrome browser. The V8 engine is written in C++ and compiles JavaScript code directly to native machine code without the need to interpret the code in real time. This feature provides Node.js with the ability for a very fast code execution. But the V8 JavaScript engine is not the only component, the Node.js architecture can be demonstrated by the figure below.

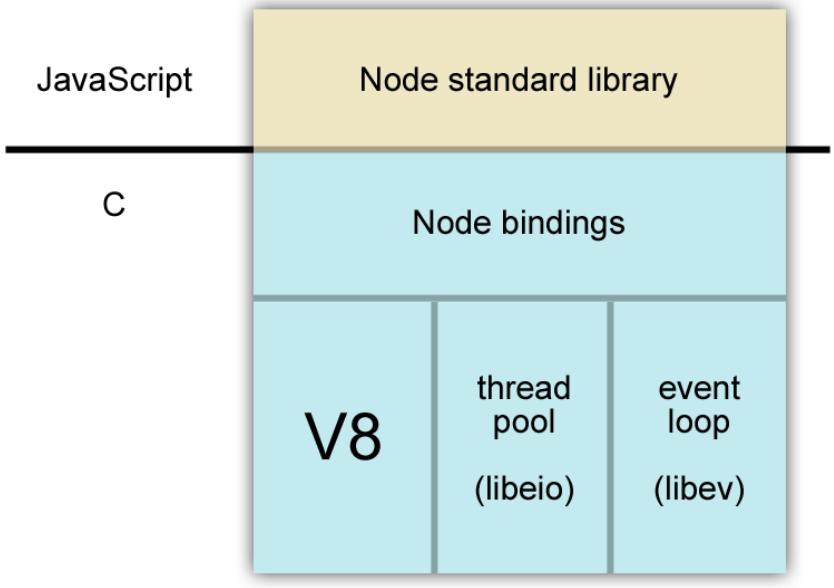

Figure 4. Node.js architecture
Copied from Tsonev (2015) [12]

As can be seen from Figure 4, the Node.js consists of several components. Namely V8 engine, libeio thread pool that is used to perform asynchronous input/output (I/O) operations and libev event loop as a foundation, Node standard library at the top and Node bindings in the middle [12]. Node standard library contains the core functionality and is almost solely written in JavaScript. Node.js bindings act as a bridge to connect the JavaScript libraries and the operating system. All the components except the top JS standard library are written in C/C++ to get a huge boost in performance.

3.1.2 Asynchronous and single-threaded

In the traditional application server model, concurrency is provided by using blocking I/O and multiple threads - one thread for every connection. Each thread must wait for I/O completion before processing the next request. The Node.js, in turn, has a single execution thread, without any context switching or I/O pending [13]. For any I/O request, processing functions called callbacks are defined that are subsequently called from the event loop, when data becomes available or something else significant occurs. The model of the event processing loop and the event handler is a common thing, this is how scripts written in JavaScript are executed in the browser [13]. It is expected that the program will quickly return control to the event loop so that the next job in the queue can be called. To illustrate the point by using an example, Ryan Dahl [13] asks what happens when executing the following code:

```
result = query (`select * from T`);
// use result
```

Listing 1. Blocking code example
Copied from Dahl (2010) [13]

Listing 1 shows the code example of a blocking programming model. And the answer to Ryan Dahl's question is that in most cases the software at this point is suspended, meaning it just doesn't do anything at all, while the database access layer sends a request to the database, which calculates the result and returns the data [13]. Depending on the complexity of the query, its execution can take a very noticeable time. This is bad, because while the thread is idle, another request may come, and if all the threads are busy, the request will simply be discarded. At the same time, context switching is not free [13]. The more threads are started, the more time the processor spends saving and restoring

their state [13]. Moreover, the execution stack of each thread takes up space in the memory [13]. And simply due to asynchronous event-driven I/O, Node.js eliminates most of this overhead, bringing in only quite a little bit of its own.

Moreover, it can be quite a challenge to implement concurrency using threads. The reason for this complexity is the need to control access to shared variables and various strategies for preventing deadlocks and contests between threads. Node.js, in turn, uses a completely different approach to achieve concurrency. Callbacks that are fired from the event loop are a much simpler concurrency model, both for understanding and for implementation.

To illustrate the need for asynchronous I/O, Ryan Dal [13] recalls the relativity of access times to different objects. Accessing the objects in memory is in order of nanoseconds and is much faster than accessing the objects from the hard drive or over the network which, in turn, is in order of seconds and milliseconds. [13] The access time to the objects that are not residing in the memory is measured by a myriad of clock cycles and can be an eternity if the client, without waiting for the page to load for two seconds, will become tired of staring at the browser window and will move to another location.

In Node.js the query mentioned above should be written as follows:

```
query (`select * from T`, function (result) {
     // use result
}):
```

Listing 2. Node.js non-blocking code example
Copied from Dahl (2010) [13]

The difference between the blocking code example demonstrated in Listing 1 and non-blocking one showed in Listing 2 is that in this example the query result is not returned as the value of the function, but is passed as a parameter to the callback function, which will be called when the result is available. Therefore, the return to the event loop occurs almost immediately, and the server can proceed to service other requests [13]. One of such requests will be the response to the query sent to the database, and then the callback function will be called. Such model of immediate return to the event loop increases the overall utilization of server resources. And it is great for the owner of the

server, but even more benefit is given to the user, for which the content of the page is available much faster.

3.1.3 Clustering

As stated above, a single Node.js server listening on a specific port runs only on a single thread. Hence, it does not take advantage of a multi-core system as it runs only on a single core. Therefore, running a single Node.js process on a multi-core system is a waste of resources. To utilize and take advantage of all resources in a multi-core system Node.js has a concept called clustering. [14]

Node.js clustering allows to easily create separate processes which all can share the same server port. For example, running Node.js HTTP server on port 80 in a cluster mode with 4 processes means that it is actually 4 separate processes all listening to port 80. And as Node.js has an asynchronous architecture with powerful V8 under the hood it has a very good performance which can be increased even further with the help of clustering. With just a couple lines of code or simply by modifying the configuration in case of use of different process management software such as PM2 the performance can be boosted by several times. However, the new problem can be with shared resources such as database which can become a bottleneck in this situation.

Node.js will load balance the requests between the processes to increase performance usually in a round-robin style. Also, it allows to achieve zero downtime as different process can crash and the requests will be forwarded to another process while this one is recovering. [14] Moreover, the zero downtime is maintained during server updates. Server update almost always requires a restart which in the case of only one process running will cause some downtime as the server needs time to restart, but with the help of clustering all the processes running in the cluster can be restarted one by one after the previous have started.

However, with all the advantages and simplicity of Node.js clustering it can be a challenge sometimes. Because to successfully utilize the concept the server must be stateless, in other words, it should not store any state in the memory. Since, it should be indifferent to which process at which time the request comes to. And making a server stateless can be a challenge. However, if it is a challenge there is a very good way

around it. Namely, in-memory shared data structure storage such as Redis or Memcached. So, the processes can utilize this shared storage to save some state and as it is located in memory it is going to be fast.

Therefore, Node.js clustering brings such powerful features such as utilization of all the server resources, load balancing and zero downtime, but, of course, with some challenges such as being stateless or storing a shared state.

3.1.4 Performance

As stated above, Node.js asynchronous and single-threaded model makes it very fast. And to confirm this statement the performance comparison and evaluation of Node.js, PHP and Python web technologies was conducted by Kai Lei, Zhi Tan and Yining Ma in 2014 [15]. The research was done by using scenario tests that were simulating realistic user behavior and benchmark tests. Benchmark test module consisted of three main parts: serving “Hello World” webpage, performing select operation from the database and calculating Fibonacci numbers’ values. [15]

The results of the first test of serving “Hello World” webpage can be seen from the following figure:

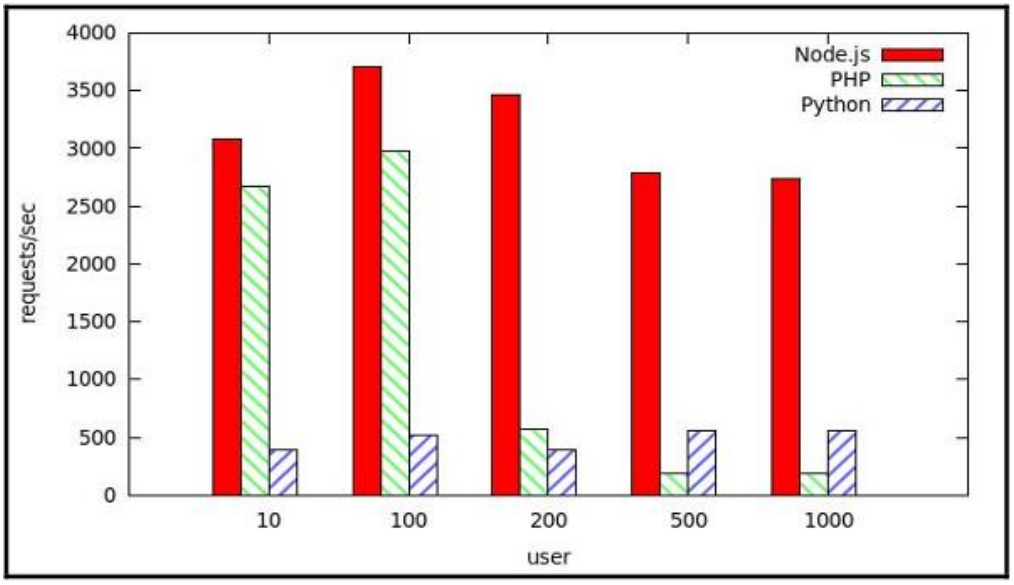

Figure 5. Mean requests per second for “Hello World” test

Copied from Kai Lei, Zhi Tan and Yining Ma (2014) [15]

The results illustrated in Figure 5 are based on two parameters: number of requests per second on the vertical axis and the number of concurrent user connections on horizontal axis. The higher the numbers the better. Therefore, based on Figure 5 it can be clearly seen that Node.js demonstrates the best performance among all three competitors. When the number of concurrent users is quite small from 10 to 100, Node.js and PHP show quite close results with around 3100 requests per second (req/s) and 2600 res/s respectively. However, when the number of users approaches 200 and then up to 1000, PHP performance drops drastically with around 200-600 req/s in comparison with Node.js which still keeps the lead with more than 2500 req/s. Python, in turn, performs steadily throughout the whole test with about 300-500 req/s.

The second benchmark test compares the performance of the chosen web technologies based on the speed of the database select operation. The results are shown in the figure below:

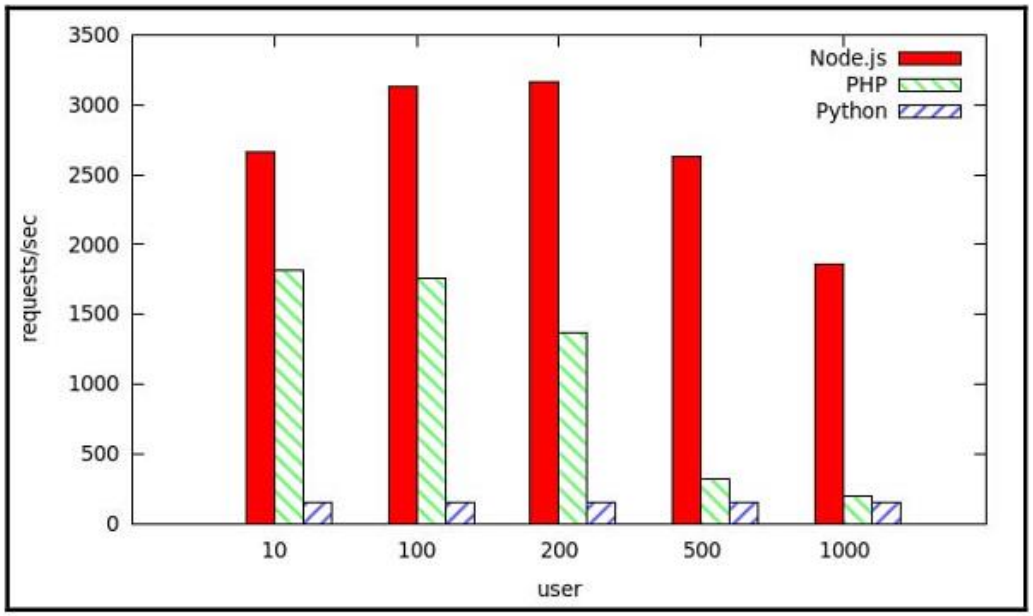

Figure 6. Mean requests per second for database select operation test
Copied from Kai Lei, Zhi Tan and Yining Ma (2014) [15]

The benchmark results that can be seen from Figure 6 are based on the same parameters as the results of “Hello World” test shown in Figure 5, namely, number of requests

per second and the number of concurrent user connections. As Figure 6 demonstrates, Node.js has the best performance with about 2600-3200 req/s when the number of concurrent user connections is less than 500 and about 1800 req/s with 1000 user connections. PHP keeps the lag of 40-60% with about 1400-1700 req/s up to 200 concurrent users. But, the gap between PHP and Node.js becomes almost two times bigger with 500 and 1000 users. Python keeps stable performance of around 200 req/s throughout the test.

The last test measures the performance of Node.js, PHP and Python-Web using the Fibonacci numbers. The test is based on the speed of executing the requests for calculating the values of the sequences of 10, 20 and 30 Fibonacci numbers. The results of the benchmark can be found in the following table:

Web development technology	Calculate value of Fibonacci	Mean requests per second [#/sec]	Mean time per request [ms]
Node.js	Fib (10/ 20/ 30)	2491.77/ 1529.4/ 58.85	0.401/ 0.654/ 16.993
Python-Web	Fib (10/ 20/ 30)	633.68/ 209.89/ 2.9	1.578/ 4.764/ 345.307
PHP	Fib (10/ 20/ 30)	2051.22/ 168.8/ 1.78	0.488/ 5.942/ 560.553

Table 1. Results for calculating 10/20/30 Fibonacci numbers
Copied from Kai Lei, Zhi Tan and Yining Ma (2014) [15]

The results demonstrated in Table 1 are compared using the number of requests per second and the time per each request. According to Table 1, the best time among chosen web development technologies has Node.js. PHP with around 2000 req/s and 0.401ms time per each request shows almost the same performance when calculating when the Fibonacci number of 10 as Node.js with almost 2500 req/s and 0.401ms time respectively. Python, in turn, lags behind almost threefold. However, when the calculations become more complex the performance gap between PHP and Node.js increases drastically and PHP starts to perform almost on the same level as Python handling only from 2 to 3 req/s with Fibonacci number of 30. Node.js, at the same time, shows the ability to handle almost 60 requests per second.

Based on the results conducted in the study, it can be clearly seen that Node.js performs better and can handle much more requests in certain moments than PHP and Python-web [15]. The tests prove that Node.js is an ideal choice for I/O intensive, high concurrency and scalable web applications with its lightness and efficiency.

3.1.5 Node Packet Manager (NPM)

NPM is the most popular JavaScript package management and distribution system, which is Node.js default package manager and the world's largest software registry [16]. Conceptually, it is similar to tools such as apt-get on Debian, rpm or yum on Redhat and Fedora, MacPorts on Mac OS X, Perl's CPAN, and PHP's PEAR. Its task is to ensure the publication and distribution of Node.js modules via the Internet using a simple command line interface. NPM allows to quickly find packages to solve a particular task, download and install them, and also manage already installed modules and necessary dependencies. All the NPM modules are stored in the "node_modules" [17]. The required dependencies for the project along with version numbers of each one are listed in the "package.json" file. Both the "node_modules" folder and "package.json" file are located in the root directory of the project. With the help of "package.json" file there is no need to upload to the server all the modules in the "node_modules" folder, it is required to upload only the "package.json" file and then run "npm install" using the command line to install all the necessary modules. Therefore, it makes Node.js projects quite lightweight as well as easy to share. [17]

3.2 Express

Express is a minimalistic and flexible web framework for Node.js applications that provides an extensive set of features and functions for mobile and web applications. It provides plenty of HTTP module services and middleware handlers, which makes the creation of a reliable and robust API a much easier and faster way than using the core node modules. Moreover, all the methods and features provided by the Express module do not hide any of the native Node.js functions. Therefore, it is possible to work with the same Node.js HTTP core objects while using Express. [18] Express consists of three main components: router, routing and middleware.

Router is the main instance of Express module which is used to specify the routes for HTTP requests, configure middleware, set template engine and etc. [18]. It can be created as follows:

```
var express = require('express');
var app = express();
```

Listing 3. Express router instance
Copied from Express [18]

As can be seen from the Listing 3, Express router object can be created by requiring the express module and calling the main “express()” function. The router instance is usually called “app” and contains all the functionality provided by Express.

Routing determines how the application should respond to a client request that was made to a specific address, named endpoint or route [18]. The definition of the route has the following structure:

app.METHOD(PATH, HANDLER)

- app is an instance of express.
- METHOD is an HTTP request method, in lowercase.
- PATH is a path on the server.
- HANDLER is the function executed when the route is matched.

Listing 4. Route definition structure
Adapted from Express [18]

Listing 4 shows that each route consists of URI or path and particular HTTP request method, for example, GET, POST and etc. Each route can have a minimum of one handler, which is executed when the particular endpoint is reached.

Express’s middleware handlers are functions that have access to the response object (res), the request object (req), and to the next middleware function in the request-response cycle of the application [18]. The next middleware function is usually denoted by the variable that is called “next”. Middleware functions can execute any code, alter the request and response objects, end the request-response cycle by sending the response

and middleware functions can call the next middleware function [18]. If the request-response cycle is not ended by the current middleware function it must always pass control to the next middleware function by calling “next()”. Otherwise, the request will hang.

All three main elements of Express can be seen in the following figure:

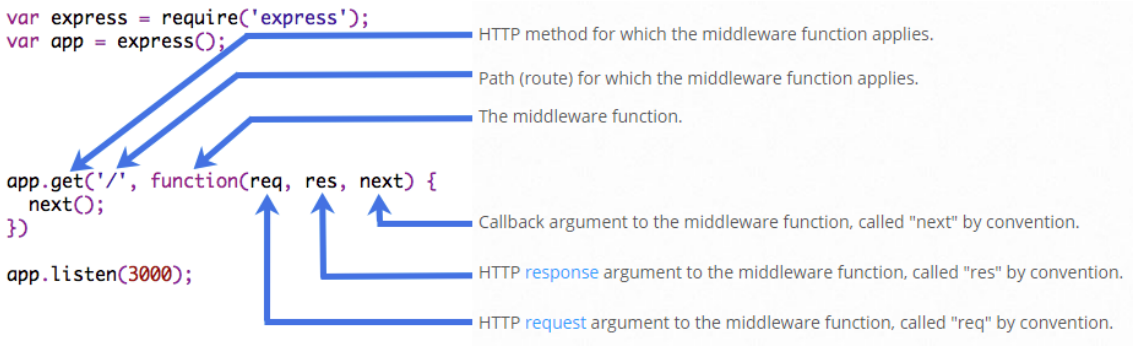

Figure 7. Express main components with middleware description
Copied from Express [18]

As demonstrated in Figure 7, the first line requires the Express module, the second line creates the main “app” instance of Express framework. The next line is the example of a route to root “/” location with middleware function that does nothing and just calls the next middleware function. The code written in the last line starts the server to listen on port 3000 for any incoming requests.

Express, at the same time, has a template engine integration which greatly helps to make the application more flexible and dynamic. The use of the template engine makes it much easier to separate application logic and views. With the template engine, it is possible to use static template files with variables. These variables are replaced by actual values at runtime by the template engine and then the template file is transformed into an HTML file that is served to the client [18]. EJS, Mustache, Pug and Jade as well, are popular template engines that are integrated to work with Express [18]. This kind of approach with runtime rendering of HTML files from static templates with support of variables makes the design of webpages a much easier and convenient process than, for example, creating the HTML files and its content by string concatenation.

3.3 MongoDB

MongoDB is a cross-platform and open-source document-oriented NoSQL database that focuses on the flexibility and scalability while, at the same time, providing high availability and performance. [19]

3.3.1 Architecture and concepts

MongoDB uses the concept of key-value pairs and the format in which the data is stored is BSON (Binary JSON) [20]. Comparing the terminology and concepts of MongoDB and any SQL Database it is possible to identify several main differences. First of all, MongoDB is conceptually the same as the usual and familiar to everybody SQL database. Inside MongoDB there may be zero or more databases, each of which is a container for other entities [20]. Secondly, a database can have zero or more "collections". The collection is so similar to the traditional "table" that they can be safely considered the same. Thirdly, collections consist of zero or more "documents". And in this case, the document can be considered as a "row". The document is a JSON object with the mandatory presence of the ObjectId that MongoDB automatically sets and controls its uniqueness, up to uniqueness around the world [20]. Fourthly, the document consists of one or more "fields", which are quite similar to "columns". Fifthly, the "indices" in MongoDB are almost identical to those in relational databases. Finally, MongoDB has a concept of “cursors” which are different from the previous five points [20]. It is important to understand that when the data is requested from from MongoDB, it returns a cursor with which the programmer can do anything like count or skip a certain number of previous records without loading the data ourselves [20]. Summarizing the points specified above, MongoDB consists of "databases", which consist of "collections". "Collections" consist of "documents". Each "document" consists of "fields". "Collections" can be indexed, which improves sampling and sorting performance. Finally, getting data from MongoDB comes down to getting a "cursor" which gives the data as needed.

The basic example of a document in MongoDB can be seen in the following listing:

```
{
          _id: ObjectId("5099803df3f4948bd2f98391"),
          name: { first: "Alan", last: "Turing" },
          birth: new Date('Jun 23, 1912'),
          death: new Date('Jun 07, 1954'),
          contribs: [ "Turing machine", "Turing test", "Turingery" ],
          views: NumberLong(1250000)
}
```

Listing 5. MongoDB document example
Copied from MongoDB [20]

As shown in Listing 5, documents in MongoDB consist of key or field – value pairs. The type of value varies and can be of any type of the BSON data types and also other documents, including arrays of documents. For example, in Listing 5 “name” field is of type document and contains another fields, in particular, “first” and “last”. Fields “birth” and “death” are of type Date. Array of strings is held in field that is called “contribs”. And “views” field has a data of NumberLong type. At the same time, the obligatory unique identifier “_id” of type ObjectID is present.

3.3.2 Dynamic schema

One of the key features of NoSQL databases and MongoDB, in particular, is dynamic schemas. Documents inside one collection can have completely different structure with fields of various types. The schema is specified not on the database level, but on the application level, meaning that new fields or documents with a different structure can be added whenever needed, even during runtime. There is no need for the developers to define the schema in advance. This feature is especially useful for improving or expanding the application, as new features can be added easily without any downtime and with no time spent for redefining the structure.

3.3.3 Replication

Replication is a way to have redundancy and get a high level of data availability. Replication is all about having several copies of data probably on different machines, which

protects the data again faults and data loss of a single machine. It also very useful for backups and recovery in disaster situations. [20]

MongoDB offers replication by the means of replica sets. Replica set can have multiple nodes that bear data and an arbiter node as an optional one. An arbiter maintains a quorum among nodes in replica set and do not store any data sets. One member of data bearing nodes is selected as primary node, while all other nodes are secondary nodes. [20] All write operations are received by the primary node and then are propagated to the secondary ones such that the data reflects the data stored in the primary node. In case of primary node failure or if it becomes unreachable, a new primary will be elected among secondary nodes. The concept of MongoDB replication can be visualized using the following figure:

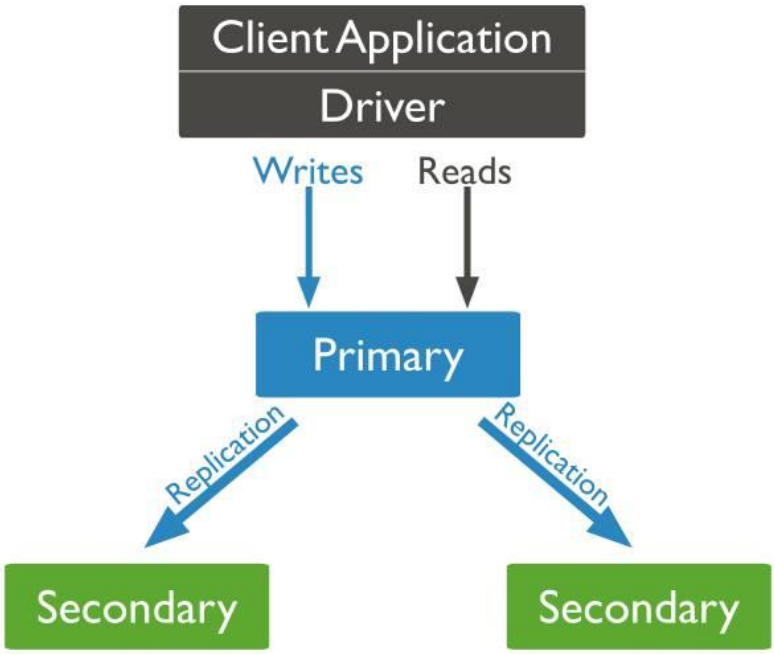

Figure 8. MongoDB replication scheme
Copied from MongoDB [20]

Based on the Figure 8, it can be seen that the server with application logic sends write and read requests to the primary node, while propagating the writes to the secondary nodes, which ends up in a synchronization of data between all nodes.

Replication adds a fault tolerance and in some cases increased read capacity to the system, which is quite essential in a production system. The system increases data availability and creates a recovery options by utilizing such MongoDB feature as replication.

3.3.4 Sharding

When the size of data becomes very large, MongoDB helps to solve this issue with the help of sharding. Scaling the system horizontally by distributing the data sets across several servers is called sharding. Thus, each machine handles only a part of the workload and a subset of data which results in improved performance compared to single server that handles all the data and workload [20]. The example of shared collection can be seen in the figure below:

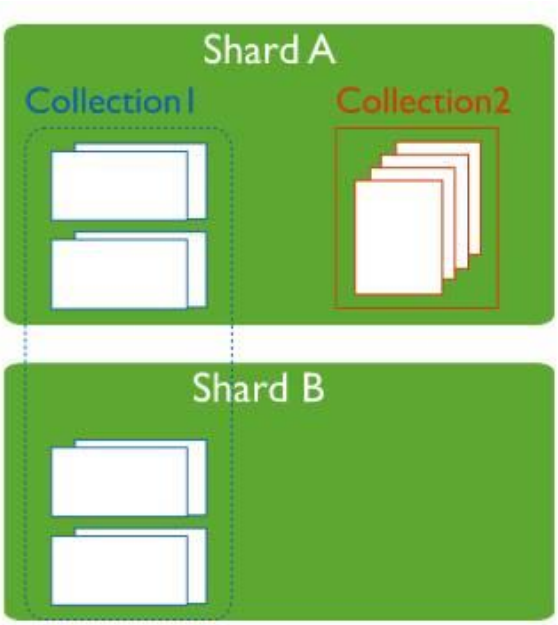

Figure 9. MongoDB sharding example
Copied from [https://docs.mongodb.com/manual]

Figure 9 demonstrates the concept of sharding. There are two collections, one of which is stored fully in one shard but the other one is divided into two shards.

Therefore, sharding provides a distributed across different machines read and write workload. Storage capacity is higher as each additional shard increased the capacity of

the whole system. Data availability becomes higher as sharded cluster is able to do partial reads and writes while some other shards could be down. [20] The price, potentially, can also be lower as getting more machines of middle performance and capacity is usually cheaper than updating already a powerful server.

3.3.5 Mongoose

Data validation and casting can be quite difficult with MongoDB, however there is Mongoose that was created to make it very straight-forward. Mongoose provides a very easy and out of the box way to model the application data by utilizing schema-based solutions. [21] Using schemas, it is possible to define at the required level what kind of data must be in the particular MongoDB document. Thus, it becomes much easier to cast, validate and boilerplate some part of business logic. Moreover, Mongoose provides methods to create and manage connections to MongoDB.

3.3.6 Pros and cons

MongoDB has a lot of positive points, such as, simple and powerful JSON-like data schema, quite flexible query language, dynamic queries, full indices support, very fast updates, efficient storage of large binary data, journaling of operations that modify data, failover and scalability support and also MongoDB can work in accordance with the MapReduce paradigm. All of these make MongoDB a very good choice for such things as high volume data feeds, ad targeting, social media monitoring, large amounts of metadata, news and different content management applications and etc. However, it also has its drawbacks, such as, lack of ‘join’ operator, which can make the organization of data quite a difficult process sometimes, lack of such concept as transaction, thus it is not possible to join several operations in one atomic action. Atomicity is guaranteed only at the level of the whole document, i.e. a partial update of the document cannot occur. Therefore, MongoDB can be a bad choice for the use in, for example, banking system as there are a lot of joined operations that must be atomic.

4 Implementation

4.1 Project Description

The project is a web application that was developed to ease the process of creation, management, registration and payment for dance tournaments. The developed platform provides a part of all required features. Nevertheless, more functionality can be easily added to the application. At the moment of writing this paper, the following interfaces were implemented:

- An admin interface to create and manage accounts for different organizations that arrange tournaments.
- An admin interface for support chat.
- An organization interface to view and update contact and business information.
- An organization interface to view and update payment credentials.
- An organization interface to create and manage created tournaments with very tailored settings.
- An organization interface to create form fields and form groups for participants to fill as a part of sign up process.
- An organization interface to create and view created email templates with the ability to insert wildcards that can later be automatically replaced by data of a corresponding tournament participant.
- An organization interface for support chat.
- A participant interface to view all available tournaments and with the ability to sign up for a chosen one.

As can be seen from the list above, the application consists of the three main parts: admin, organization and participant interfaces.

4.2 Environment setup

The application was developed on a desktop running Windows 10 operating system. Apart from the PC and OS the implementation required a development environment to

be set up. The development environment consists of different software programs and tools to work with them. Thereby, the following software was installed:

Node.js

Node.js is the essential part for the development process of this project as it provides the runtime environment for JavaScript code execution. It can be downloaded from the official website (https://nodejs.org/en/download/). After the installation, the easiest way to check if it is working properly, is to run `node -v` command in the terminal. The command prints the installed version of Node.js. The 6.3.1 version was used during the development.

MongoDB

MongoDB is the second most important part of the development environment because the application requires a database to store and query various data. It can be installed from the official website (https://www.mongodb.com/download-center). After the installation, MongoDB server can be started using `mongod` command from the terminal. The 3.2.3 version was used during the development.

WebStorm

WebStorm is a very powerful Integrated Development Environment (IDE) for JavaScript development. It has a built-in support for Node.js projects. As well as code completion, powerful navigation features, on-the-fly error detection, and refactoring features, it has a friendly interface for debugging Node.js applications.

Robomongo

Robomongo is a MongoDB management tool with a very friendly user interface and native MongoDB shell support.

SourceTree

SourceTree is a very powerful Git GUI with great and simple interface.

4.3 Project structure and implementation

The project structure is well defined and components are separated correspondingly to their roles. The structure overview can be seen in the following figure:

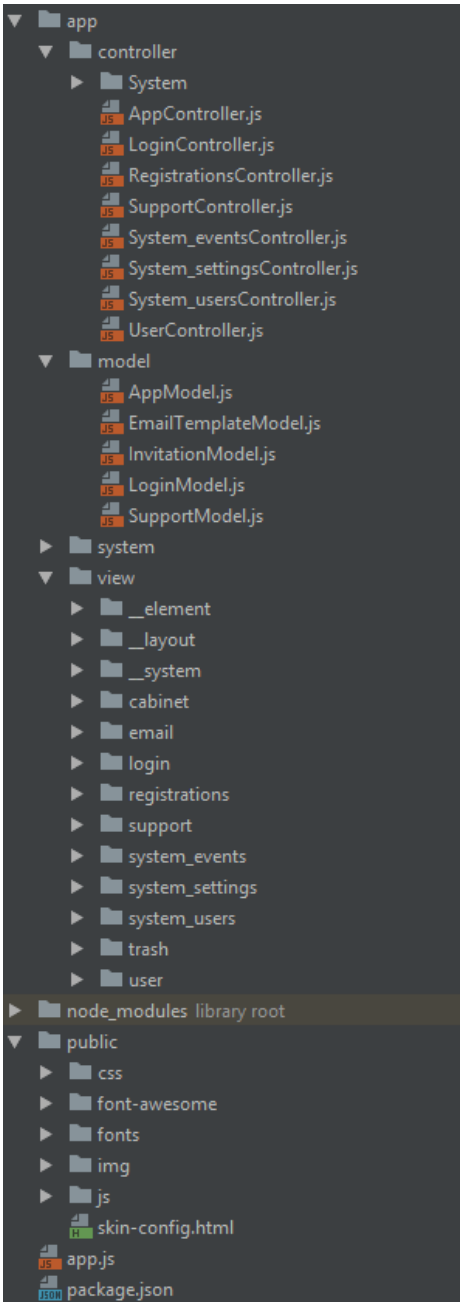

Figure 10. Project structure overview of admin interface

Figure 10 demonstrates the structure of the application. The entry point for the application is the “app.js” file. It contains all the initialization, configuration, routing and server

starting logic. The business logic is located under the “controller” directory. All the models corresponding to the database collections are located inside the “model” directory. The “view” directory contains the representation components that are shown to the user of the application. All the images, CSS, fonts, images and JS code used in the front-end lie under the “public” directory. Different utility functions are located in “system” directory. And the “package.json” file contains the list of all dependencies that are used in the project.

As was stated above, “app.js” is the main and entry file in the application. First of all, it reads and initializes all the models in the model directory. Then it requires and initializes instances of all controllers that are located in controller folder. Then the instance of “express” is created and the connection to the database using the “mongoose” module is established. After that all the required configurations are made for the “express” instance. Then, with the help of a module called “passport” a simple session authentication system is created. Following, the handlers for the GET and POST requests and separately for “/login” route are created using the “express” instance. And finally, the “express” server is started on the port that is specified in the configuration.

The next most important part of the application is business logic. The controllers that are defined in “controller” directory, are responsible for all business logic that happens in the application. Each controller has the same structure and must extend the main controller that is called “AppController”. It contains the logic for parsing and preparing the request information to be easily accessible by each of child components. As well as having the initialization and preparation logic, “AppController” has common methods that are often used by the derived classes.

As was explored above, the model directory contains the models that are an abstraction over the corresponding MongoDB collections. Every model must extend the class that is called “AppModel”. The parent class contains the logic for instantiation of the Mongoose model with the corresponding schema. Moreover, the common create, read, update, delete (CRUD) methods to work with the collection are as well defined in “AppModel”.

The views are the last part of the application. Each view is a EJS template with defined HTML markup and JavaScript code to insert the data into the template. Some EJS files contain a small component created for a particular page, while a part of views defines a

common layout that is used throughout the application. Layout, body and all the required EJS templates are combined into a final view before sending to the client. In this way, a user of the application can see a complete HTML webpage with necessary components.

4.3.1 Request-response flow

The request-response flow in the application described in this paper is based on a custom framework. It was decided to create a custom framework on top of standard “express” router to have a better control and flexibility. Thus, every GET and POST request is intercepted by the corresponding handler and then the URL is parsed to determine the correct controller and its method to handle the request. The URL must be in the form:

```
/{controller name}/{method}
```

Listing 6. URL template

As listing 6 indicates, every request URL must contain the name of the controller and the correct method. Hence, after parsing the URL, the list of all controllers is checked if it has the requested controller. If the controller is not present in the list, a 404 status will be send to the client. On the other hand, if the controller is present, the request and response objects are prepared and added to the instance of the controller and then the requested method is invoked with either GET or POST parameters. Therefore, inside the invoked method there is an access to request, response objects and the requested parameters. As well as request and response objects, the default model with the same name as the controller is available in the scope of the method. During the method execution all the required business logic, such as database read or writes and data processing, is carried out. At the end of the execution, it is possible to send the response with just data or render the necessary page or a particular element with data using the tailored view system and send back HTML markup.

Thereby, the created framework allows for a very easy, flexible and transparent development process. It is very convenient to add more controllers, models and views. Using this framework, the application can be easily expanded with more pages, information and functionality.

4.3.2 Controllers

This subchapter takes a closer look and describes the role of each controller presented in the biggest part of the project, which is organization interface.

Cabinet Controller

Cabinet controller is responsible for saving and retrieving organization information such as payment credentials, contact and business info. There are several POST and GET methods presented in this controller. First of all, a base “index” method:

```
index() {
            this.redirect("/cabinet/dashboard");
}
```

Listing 7. “Index” method of cabinet controller

As can be seen from listing 7, the “index” method redirects to the relative URI “cabinet/dashboard”. The redirect method is declared inside “AppController” and is accessible as each controller extends “AppController”. The method calls the redirect function of response object. The redirected URI invokes invokes a “dashboard” method of the same controller that is demonstrated below:

```
dashboard(){
            this.render({ __title : "Dashboard"});
}
```

Listing 8. “dashboard” method of cabinet controller

Listing 8 shows the implementation of “dashboard” method of cabinet controller. The method calls “render” function on the instance of the class. “Render” method that is defined in “AppController” prepares the necessary options and renders the view that corresponds to the “dashboard” method. The view contains a dashboard interface with statistics overview. Secondly, there are three methods for saving organization information: “saveContactInfo”, “saveBusinessInfo” and “savePaymentCredentials”. All of them have the same structure:

```
saveContactInfo(){
      this.loadModel("User");
      this.User.update(
            {_id : this.user._id},
            {"$set" : {"contact_info" : this.request_data}},
            (err) => {
                  this.sendStatus(err ? 200 : 500);
            });
}
```

Listing 9. “saveContactInfo” method of cabinet controller

Listing 9 demonstrates the code snippet of “saveContactInfo” method. The first line in the method calls the “loadModel” method of “AppController” to prepare the requested model and add it to the instance of the class. The second line calls the “update” method of the loaded “User” model. The first argument is the query, meaning what documents in the collection should be updated. The second argument contains the update payload. And the third and last argument is a callback function that will be invoked once the update either succeeds or returns an error. The callback checks if the error is returned and send the corresponding status to the client. “SendStatus” method is an abstraction above the “sendStatus” method of response object created by “express”. Lastly, to display the saved data, two similar methods are used: “info” method to get and render the contact and business information and “paymentCredentials” method to render the view with payment credentials data. Both methods contain only one line:

```
paymentCredentials(){
      this.findOneAndRender(
            {_id : this.user._id},
            "User",
            { __title : "Payment credentials"},
            "user"
      );
}
```

Listing 10. “paymentCredentials” method of cabinet controller

The code example demonstrated in listing 10 shows the implementation of “paymentCredentials” method. As can be seen, the method only calls the “findOneAndRender” method that is defined in “AppController”. The method is and abstraction above querying the data from database and rendering the particular view with that data. The first argument is the database query. The second one is the model name that should be queried. The next argument contains the additional data that will be passed to view. And the name of a view is passed as the last argument to “findOneAndRender” method.

Form fields Controller

Form fields controller has the logic for displaying and saving form fields created by the user, as well as forming and displaying groups of the created form fields. The created form fields and form field groups are used in tournaments and email templates creation. The following methods build up the logic of the controller. “Index” method redirects to the method called “show” that renders the interface for creating and displaying form fields. The creation of the form fields is supported by the “save” method which prepares and saves the desired form field. And, of course, there is “remove” method to delete the created form field. The most part of the logic in the controller is occupied with handling form field groups. First of all, the “groups” method finds all the previously created form groups and form fields and renders the view for creating groups from the created form fields and displaying the overview of the found ones. With the help of “getGroup” method it is possible to get the view with detailed information of the requested form group. To create and remove groups the following methods are necessary: “saveGroup” and “removeGroup” correspondingly.

Email Controller

Email controller is responsible for saving and displaying email templates that can created using a modular editor. It is also possible to insert wildcards to the template that were formed from the created form fields. The first and basic method is “index”. It redirects to “myTemplates” method to render the view that contains the list of all email templates that were created by the user. From this view it is possible to either delete the template with the help of “removeTemplate” method or edit the template. If the edit option is chosen, the page will be redirected to the “templateEditor” method that renders the view with the editor and using “getTemplate” method the data of the chosen template is loaded to the client. The editor view also contains a list of wildcards that can be used in the creation of various templates. To get the list of wildcards, “getWildcards” method is used. And finally, new template or the edited existing template can be saved to the database using the “saveTemplate” method.

Login Controller

Login controller has only two methods. “Index” that renders the login page and “logout” that logs out the user and redirects to login page. The login, session and logout functionalities are provided via NPM packages: “passport”, “express-session” and “passport-local”. On the login page after entering username and password a POST call to “login” endpoint is made. The endpoint is created using “express” and “passport”, no controller is taking part in authorization. The handler that is registered on “login” endpoint checks username and password and in case of correct credentials login session is instantiated. The logout is made using “passport” method “logout” that is attached to request object on subsequent requests.

Support Controller

Support controller is responsible for rendering support page, saving and getting messages. The main “index” method finds all the conversations in the database and renders the page with the list of conversations. The rendered page invokes “messages” method with the id of the first conversation. The method queries the database for all messages in the particular conversation and returns a rendered list that is then displayed in the browser. The last method presented in the support controller is “saveNewMessage”. As can be guessed from the name, it is responsible for saving messages to the database.

Tournament Controller

Tournament controller contains the logic for finding, editing, creating and removing tournaments. The base “index” method redirects to “manage” method which renders the view for tournaments list with search fields. When the page opened on the client, “get” method is invoked. By default, the method finds all tournaments created by the user and returns a rendered list to the client. The same method is used with different parameters when search for particular tournaments suing search fields. The user can click on any tournament in the list to get the detailed information. The information is requested with the help of “detail” method. It gets the data and then sends to the client a rendered view of detailed information on the requested tournament. The view with details has four buttons to manage the selected tournament. It is possible to toggle active status of the tournament using “toggleActive” method, to toggle tournament visibility with the help of “toggleVisible” method. The tournament can be deleted using “remove” method. The last button is responsible for redirecting to the editor page to edit the particular tournament. The editor

page is used for both creating and editing the tournaments. “Create” method is responsible for rendering the tournament editor. If the id of the tournament is passed to the method, the view is rendered with the information of the selected tournament. Overwise, the view is rendered with empty fields. The last but not least, is “save” method which is responsible for either saving a created tournament or the edited one.

4.3.3 Models

Each model extends the main “AppModel” class. The parent class has method abstractions for each database operation, such as, find, findOne, distinct, create, insert, update and etc. Therefore, the controllers make queries to the database only using the methods defined in “AppModel”. For example, the method responsible for inserting the documents into the database can be seen below:

```
insert(documents, callback){
      let data = documents;

      if(!Array.isArray(data)){
            data = [data];
      }

      this._model.collection.insert(data, callback);
}
```

Listing 11. “insert” method of main model class

Listing 11 demonstrates one of the abstractions presented in “AppModel” class. As can be seen the method accepts two parameters: a document or an array of documents that needs to be inserted into the database and a callback that will be invoked with the result data once the operation ends. First couple of lines in the method check if the provided documents variable is a single document or an array and if the variable is not an array converts it to the array with a single element. The last line invokes the “insert” method with the provided data and callback on the corresponding collection using a mongoose model. Mongoose model is created using the “_initiate” method of “AppModel”. The body of the initialization method is, as follows:

```
_initiate(schema, model_name){
      this.collection_name = model_name;
      this._model = mongoose.model(this.collection_name, new Schema(schema));
}
```

Listing 12. “_initiate” method of main model class

Listing 12 shows the implementation of “_initiate” method of the main model class. The method takes mongoose schema and model name as arguments. The first line assigns the model name to the class variable called “collection_name”. The second and last line is responsible for initiating the mongoose model using the provided schema and model name. The initiated model is then stored in the “_model” property of the class. The provided schema can be defined in two places. First of all, if a model does not require any specific methods then the schema must be defined in “Schemas” file that is located in “system” directory. And in this case, there is no need to create a separate class for the model. On the other hand, if a model does require some methods that are exclusively used with that model, it must be defined as a separate class that extends the “AppModel” class. If the model falls under the second category then the schema must be defined in the “schema” property of the class. The schema for one of the models can be seen below:

```
{
      "username" : String,
      "password" : String,
      "contact_info" : {
            "phone" : {type : String, default : ""},
            "email" : {type : String, default : ""},
            "surname" : {type : String, default : ""},
            "name" : {type : String, default : ""}
      },
      "business_info" : {
            "type" : {type : String, default : ""},
            "address" : {type : String, default : ""},
            "number" : {type : String, default : ""},
            "orgname" : {type : String, default : ""}
      },
      "role" : String,
      "approved" : {type : Boolean, default : false},
      "finance" : {type : Object},
      "active" : {type : Boolean, default : false}
}
```

Listing 13. The schema for user model

The schema shown in listing 13 belongs to user model. It describes every field, its type and default value. First of all, a user has username and password. Secondly, the user has a role such as, admin, organization or client. Then, the user can be approved and activated, therefore statuses approved and active are required. And, finally, there is personal information: contact, business and payment information that is managed in cabinet controller.

4.3.4 Views

The views in the project are created using EJS template engine. There are several types of views in the application described in this paper. First of all, general views, such as “layout.ejs”, “body.ejs”, “javascript_global.ejs” and etc. Secondly, views associated with the particular controller and its method, for example, in the directory called “cabinet” there is the view called “info.ejs” which is the default view for the “info” method in cabinet controller. And finally, there are separate elements that are used in views associated with controllers.

The views have the following general structure and flow. Some controller renders a particular view. Each view defines the layout that the view must be used with using the following EJS syntax:

```
<% layout('./__layout/layout') -%>
```

Listing 13. EJS layout definition

Listing 13 shows the code for defining the layout for a view. The “layout.ejs” file described above, includes “styles_global.ejs” which contains all CSS used in the application globally. Based on the authentication status, the “layout.ejs” file includes either “body.ejs” or “body_no_auth.ejs” as well. Both views, in their turn, include “javascript_global.ejs” file that defines all JavaScript libraries that are used in the project, such as “jquery”, “toastr”, “bootstrap” and etc. And “body.ejs” view also includes “left_menu.ejs” which creates the navigation menu.

Most of the lines in views are occupied with raw HTML markup and only a part has EJS syntax for rendering some dynamic data. The code sample for a small part of a support chat view can be seen below:

```
<div class="users-list">
     <% for(let i = subjects.length-1; i >= 0; i--){ %>
          <div class="chat-user subject" data-subject="<%= subjects[i] %>">
               <div class="chat-user-name">
                    <%= subjects[i] %>
               </div>
          </div>
     <% } %>
</div>
```

Listing 14. EJS code sample for a list of conversations

Listing 14 demonstrates a way of inserting data into HTML using EJS syntax. The code sample shown in listing 14 creates a list of all conversations in support chat. As can be seen EJS code is wrapped in “<% %>” tags that are called “scriptlets”. Inside there is a loop over the conversation subjects. All the HTML that is inside the loop will be created for each iteration. Inside the loop the value of each conversation subject is inserted into two places using “<%= %>” tags that output the value into the template. In the same way all the dynamic content is created in the project.

5 Results

The result of this thesis was a working web application with a list of features. Although, it is not a finished product, it has a base structure and a number of working parts and components. At the same time, as the main framework is fully functional, more features and functionality can be easily added to the application. The following screenshots demonstrate several parts of the application:

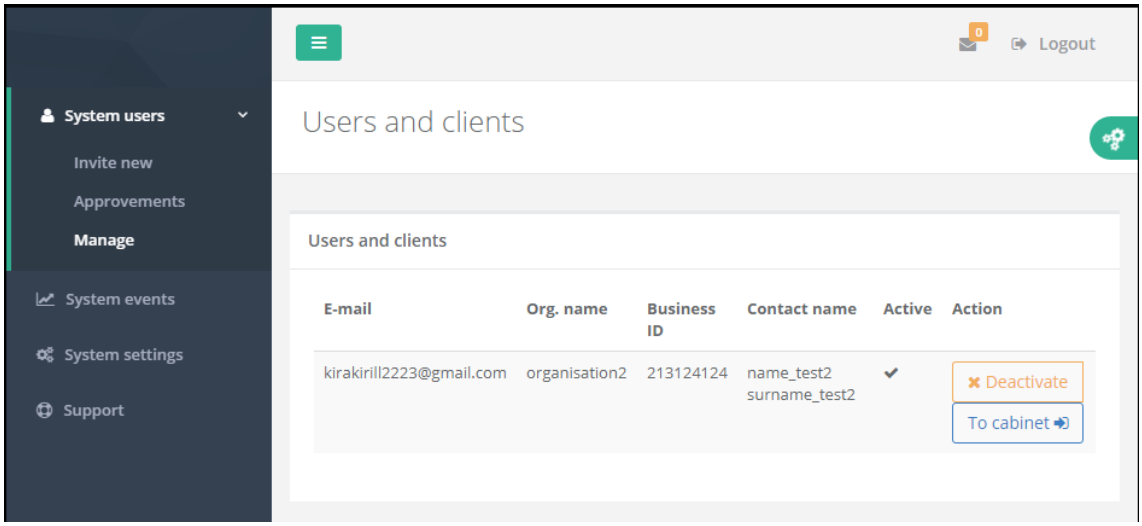

Figure 11. Admin interface to manage accounts

Figure 11 illustrates the admin interface for management of the accounts for different organizations. The table contains the main information about the organizations and it is possible to deactivate and reactivate the accounts at any time.

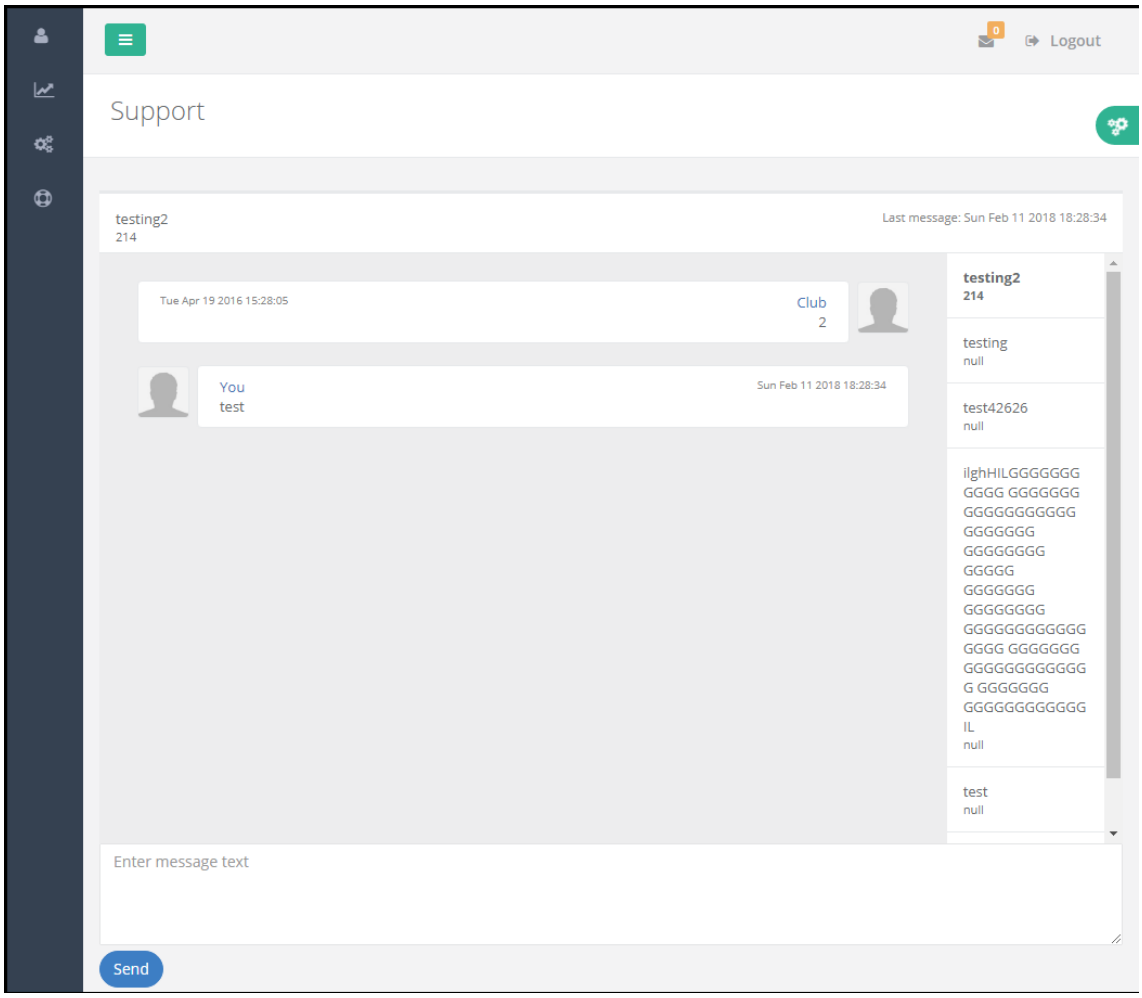

Figure 12. Support chat interface

Figure 12 demonstrates the support chat interface. This module is accessible by admins and organizations. On this page there is a list of messages of a selected conversation with a certain organization. The date of each message and specifically of the last one in the current conversation can be seen on the figure. The list of all available conversations with corresponding titles and the name of the organization the initiated the discussion is shown on the right of the figure.

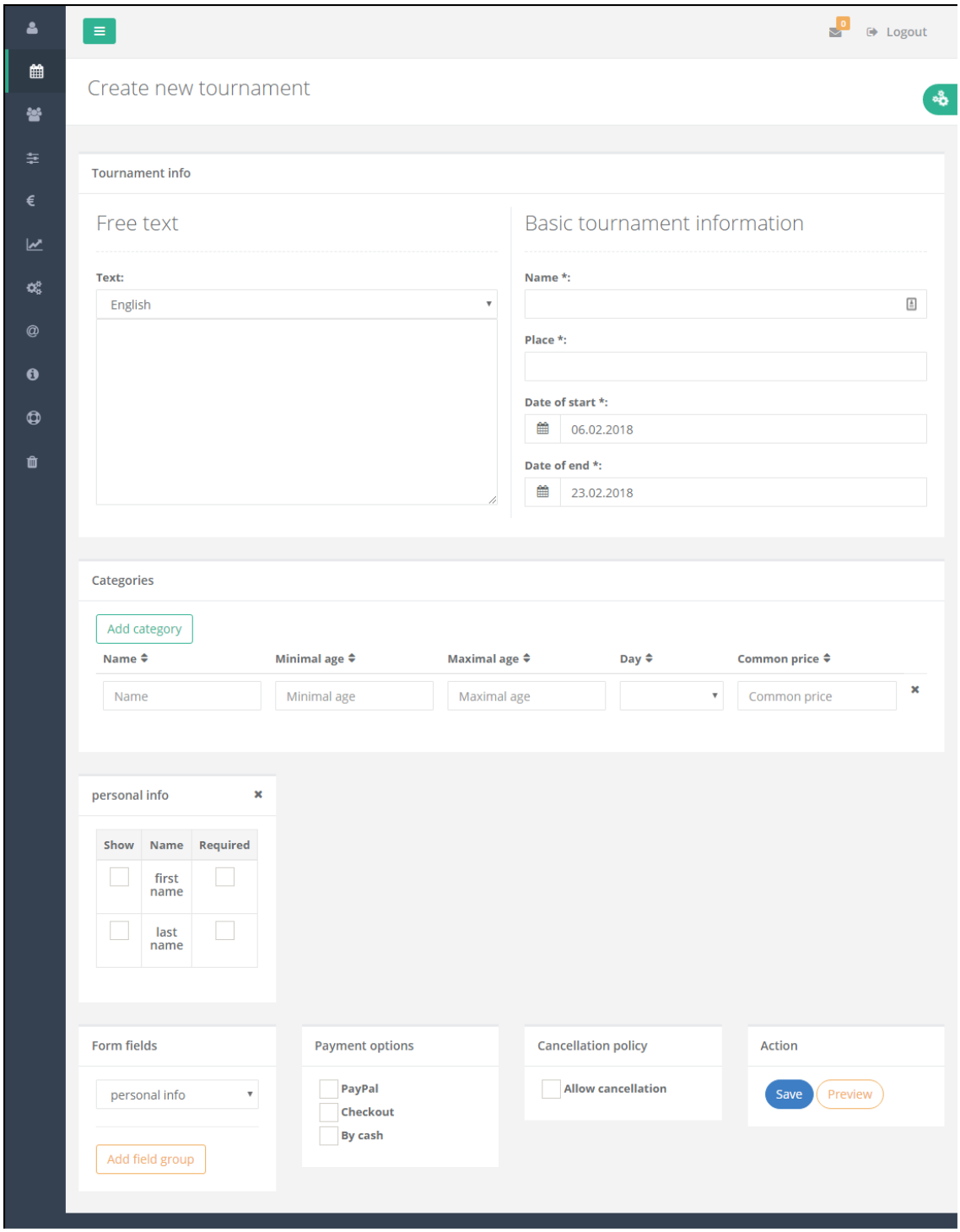

Figure 13. Interface to create new tournament

Figure 13 shows the organization interface to create a new tournament. As can be seen from the figure, it is possible to specify the basic tournament information such as name, place and dates of the tournament. Different categories with various settings can be added as well. It is also possible to specify the fields that all the participants are required

to fill during the registration. The payment options are also available for selection. And after saving the built tournament it will be available for sign up in participant's interface.

The paper's main goal in developing a working web application was successfully reached. The technologies used in the project were thoroughly studied and described. Good code structure and organization were introduced. A core framework with easy, understandable application flow and a space for future improvements was developed.

6 Discussion

The application was developed using MongoDB and Node.js. Server-side rendering with raw CSS and JS were used to create user interfaces. The code was separated into different modules, such as view, models and controllers. Therefore, leaving the code clean and clear and very easy to maintain and extend in the future.

However, as any project, the developed product it is not ideal. The application lacks automated testing, such as unit, acceptance and front-end testing with screenshots. Of course, the application was continuously tested manually, but it does not eliminate all the errors and bugs. Therefore, some unexpected errors or behavior might occur while using the application.

There are no front-end frameworks in the project, such as React, Angular or Vue.js. On one hand, as the project described in the paper is not a Single Page Application (SPA), it does not have a long initial loading time. Every page is requested from the server in a separate request and therefore, there is no need to load all the resources at once. On the other hand, SPA is a very popular approach nowadays and it has a lot of advantages. First of all, it is possible to move routing from the server to the front-end and use a server only as a data source. Then, it is very easy to modularize the components with the help of front-end frameworks. At the same time, it could be a good idea to move away from simple CRUD HTTP requests and utilize such modern technologies as GraphQL with the help of Apollo Client in the front-end. GraphQL makes it much easier to request only the necessary data from the same endpoint without the need to create dozens of HTTP endpoints for each kind of data and needs.

Hence, despite the fact that the project resulted in the clean, organized and functional prototype, there is always multiple different technologies and approaches that could be used, as well as a very vast space for improvements.

7 Conclusion

The focus of this thesis was to study the technologies involved in the Full Stack JavaScript development and create a working web application based on the studied topics. The research took a large amount of time, but as a result, all the required technologies, such as Node.js, MongoDB, Express were thoroughly analyzed. Their advantages and disadvantages were identified and compared to other technologies that are as well used in web development.

After a detailed research of a theoretical aspect of the project, the practical part was carried out. Careful architecture design and then actual implementation of the application resulted in the working prototype of a platform for management of dance tournaments. The purpose of the development was to show the implementation of the studied topics and a successful combination of a separately working parts. The project can be further improved with a number of features and transformed into a production ready system.

After a detailed research and an actual implementation of the project using Node.js and MongoDB, it is possible to make several conclusions. First of all, it requires the knowledge of only one programming language, which can be essential for small companies as it allows to hire only one developer. Secondly, it is a very scalable environment that can be used for a rapid development and a quick web server set up. However, the technologies can be also applied for highload projects with the help of MongoDB sharding, replication and Node.js clustering. Thirdly, there is a huge JavaScript community that can be of a very great help. And finally, it can be concluded that the Node.js, MongoDB and Express bundle is one of the best choices for a modern web application development.

References

1 Codesido I. What is front-end development? [online]. Guardian official website; 28 September 2009
URL: https://www.theguardian.com/help/insideguardian/2009/sep/28/blogpost.
Accessed: 6 November 2016

2 World Wide Web Consortium. What is CSS? [online]. W3 official website.
URL: https://www.w3.org/standards/webdesign/htmlcss#whatcss.
Accessed: 6 November 2016

3 Steven M. Schafer. HTML, XHTML, and CSS Bible, 5th Edition. Crosspoint Boulevard, IN: Wiley Publishing; 2010

4 JavaScript – Overview [online]. TutorialsPoint official website.
URL: https://www.tutorialspoint.com/javascript/javascript_overview.htm.
Accessed: 6 November 2016

5 What is a web server? [online]. MDN official website.
URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server.
Accessed: 6 November 2016

6 September 2016 Web Server Survey [online]. Netcraft official website; 19 September 2016
URL: https://news.netcraft.com/archives/2016/09/19/september-2016-web-server-survey.html
Accessed: 6 November 2016

7 Server-side web frameworks [online]. MSDN official website.
URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks
Accessed: 12 February 2017

8 Business Layer Guidelines [online]. MSDN official website.
URL: https://msdn.microsoft.com/en-in/library/ee658103.aspx
Accessed: 11 February 2017

9 NoSQL vs SQL [online]. MSDN official website.
URL: https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql
Accessed: 12 February 2017

10 Leavitt N. Will NoSQL Databases Live Up to Their Promise? Computer. February 2010, 43(2).

11 Node.js official website [online].
URL: https://nodejs.org
Accessed: 15 April 2017

12 Krasimir Tsonev. Node.js By Example. Birmingham, United Kingdom: Packt Publishing Limited; 25 May 2015

13 Cinco de NodeJS — May's BayJax Celebrates Server-Side JavaScript with Ryan Dahl, Elijah Insua, and Dav Glass. YUI; 27 April 2010

14 Node.js official API website [online].
URL: https://nodejs.org/api/cluster.html
Accessed: 14 May 2017

15 Kai Lei, Yining Ma, Zhi Tan. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python and Node.js. Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on; 2014

16 NPM official website [online].
URL: https://www.npmjs.com
Accessed: 14 May 2017

17 Simon Holmes. Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications; 2015

18 Express official website [online].
URL: http://expressjs.com
Accessed: 21 May 2017

19 MongoDB official website [online].
URL: https://www.mongodb.com
Accessed: 15 July 2017

20 MongoDB official documentation [online].
URL: https://docs.mongodb.com/
Accessed: 16 July 2017

21 Mongoose official website [online].
URL: http://mongoosejs.com/
Accessed: 21 July 2017