WEB APPLICATION FOR THE DETECTION SYSTEM OF RISK SITUATIONS AT THE RAILWAY CROSSING

Jan Bachorec
Master Programme (2), FEEC BUT
E-mail: xbacho01@stud.feec.vutbr.cz

Supervised by: Zdeněk Martinásek
E-mail: martinasek@feec.vutbr.cz

Abstract: The paper describes a design and an implementation of the complex server application including the graphical user interface. The application is designed to detect dangerous situations at the rail crossing. The final own implementation consists of a server application and a database (full stack approach). The server application is based on .NET Core. Its purpose is to receive, process and securely store data from autonomous cameras into database. The database is based on the Cassandra database system. Graphical user interface is an interactive website implemented with Razor Pages technology.

Keywords: web application, visualisation, database systems, API services

1 INTRODUCTION

In recent years there has been an increase in traffic intensity. According to statistics from the Ministry of Transport [1], traffic on most types of roads increased by an average of almost 12 % between 2010 and 2016. Unfortunately, the higher traffic density is associated with an increase in the number of traffic accidents, as can be seen from police statistics [2]. Accidents at the railway crossing, as reported in the materials of the BESIP [3] organization, do not have a dominant share in the absolute number of accidents. On the other hand, when such an accident occurs, it tends to be very serious and often involves injuries and loss of life. The mentioned statistics clearly show the need to introduce additional safety mechanisms to help reduce the number of dangerous situations at level crossings, thereby reducing damage to health and property.

In this paper, I am researching, developing and implementing a complex server solution for the autonomous system for detecting dangerous situations at railway crossings (ADEROS). The main purpose of the server solution is to collect and store data from individual camera modules monitoring railway crossings. The graphical user interface consists of the interactive web page and is intended to provide an overview of detected situations. Operator can analyze these situations.

2 OWN DESIGN AND IMPLEMENTATION OF THE SERVER SOLUTION

The system for autonomous detection of risk situations in transport consists of several basic components. The server solution, whose design and implementation I describe in this paper, consists of a data gathering and processing service. Collected data are stored in the database. Cameras autonomously detecting different traffic situations are used as a data source for the server solution. Interaction between the system operator and the system itself is provided by the own interactive web application. Its development is discussed in the following chapter. Figure 1 shows the complete architecture design of the system. The following text contains more detailed descriptions of the basic components of the server solution of the system for autonomous detection of risk situations at a railway crossing.
The created server service provides an application interface for receiving messages from the cameras. And it also allows to process and store the messages into a database. The technology chosen for the implementation of the server service is the .NET Core platform and the C# programming language. With the help of these tools, I have implemented the REST API (Application Programming Interface) required to receive both event and status messages from the camera modules. Their detailed description is beyond the scope of this article, but it is necessary to describe communication interface of cameras. It consists of an HTTP module that allows sending camera status and event data in JSON (JavaScript Object Notation) format to the server solution. It also consists of an SSH (Secure Shell) server, which can be used to remote management of the cameras.

The created application consists of API controllers that are used to serve HTTP (Hypertext Transfer Protocol) requests from the cameras, containing information about their status and also about detected events. At the same time, these controllers, using the technique called DI (Dependency Injection), use other supporting classes and their methods to parse, process and store the data in the database.

To work with the database server, the CassandraCSharpDriver version 3.16.1 is used. The program contains a namespace with classes that use this driver and provide communication with the database and correct mapping of data to the appropriate columns of the database tables. These classes include objects representing database cluster connections and sessions. In this case, the DI is used in such a way that only one instance of these objects is worked with during the program run in order to follow the recommendations for proper use of the CassandraCSharpDriver.

Microsoft IIS (Internet Information Services) was chosen as the web server technology to run the .NET Core server service. As mentioned in the documentation, it is a server bundled with the Windows operating system and Windows Server. More specifically, the current version of IIS 10.0 comes with the Windows 10 operating system and the Fall Creators Update package installed, as well as the Window Server 2016 v. 1709 server operating system.

Various database systems were considered for data storage. When deciding between them, I was guided by the requirement to store a large amount of data in real time. Furthermore, I preferred to use open source software. It was also necessary to consider the planned extension of the application. For example, it is planned to include more advanced statistical data processing. I decided to chose the Elassandra tool for data storage. It is a software combining the Cassandra NoSQL (Not only Structured Query Language) database system with the Elasticsearch search tool. Cassandra can ef-
ficiently handle a large number of data write operations if the schema design is appropriate, while providing considerable flexibility in data model design options. For example, through the support of user-defined data types. In the context of this work, Cassandra is used as a repository for detected events and status messages, as well as other information necessary for running developed systems. Elasticsearch provides a secondary index over tables.

3 OWN GRAPHICAL USER INTERFACE IMPLEMENTATION

The graphical interface is used to clearly display the detected situations to the ADEROS operator. I have created my own interactive web application using Razor Pages technology. This technology is part of the .NET Core framework, also used for the development of the server service described in previous section. In order to separate working with the database and other necessary services from the user interface itself, I created a REST API service for this purpose, also built on .NET Core and implementing all database operations.

Web application, whose interface is shown in the picture 2, contains four basic modules from the user’s point of view. Within the camera module there is an overview of all available cameras in the system. The purpose here is to inform the operator if the camera is connected and communicating properly. It is also possible to click on a camera to view detailed information about its status and also, for example, to change its name in the system. The alert module informs about the latest detected risks. Again, you can find both the summary information about the latest alerts from all cameras and, after opening the appropriate alert, its detail. Within the events tab it is possible to view all recorded events. Even those that are not classified as alerts. The filtering of the displayed events uses name of the source camera and the time range of event detection. Administration allows, for example, to manage user accounts within the system.

The web application is secured against unauthorized access. A REST API service is used for user authentication. If the user is registered in the system, the API service returns a JSON Web Token (JWT) token, based on which the web application issues a Cookie to the user’s browser, authorizing the user to access individual web pages.

![User interface of the own interactive web application.](image)

Figure 2: User interface of the own interactive web application.

API service is built on .NET Core and its task is to provide the necessary data to the web application via REST API so that information can be displayed to the user. The service contains both HTTP
controllers to handle requests from the web application, and classes that implement all supported operations. The interconnection between the controllers and the mentioned classes is implemented using a dependency injection technique. Working with tables in the Cassandra database is implemented similarly to the server service described above, using the CassandraCSharp driver. Communication between the web application and the API service is implemented using HTTP messages. Data is transferred in JSON objects within the body of the exchanged messages.

The service controllers are secured to prevent unauthorized access to the system. The service has implemented methods for authentication, in which it issues a JWT token to an authorized user based on knowledge of the login credentials. This is a JSON object signed with the service’s secret key. This token contains information about the user and his role in the system. HTTP requests are then handled only if they contain a valid token.

4 CONCLUSION

This paper describes the research and the development of the server application for the system for detecting risky situations at a railway crossing. It is a complex system that allows to clearly display and analyze data obtained by cameras with autonomous image recognition. As a part of the actual design and development, I first had to implement a database schema design using Cassandra technology. I designed the scheme with an attention to high performance even in the case of connecting a large number of camera modules. The next step was to create a server service for collecting, processing and storing the data into the created database. I used the .NET Core platform to implement it. To enable interaction between the operator and the ADEROS system, I designed and implemented the interactive web application, built on Razor Pages technology. To provide this application with data, I also implemented the REST API service through which data is obtained from the database. The implementation of all components of the system has been tested and is fully functional. The advantage of the technologies used is the easy extensibility of the system if needed.

ACKNOWLEDGEMENTS

This paper was created with the support of the Ministry of Industry and Trade project “Autonomous system for detection of risk situations in traffic based on image sequence analysis” with reg.no. FV40372.

REFERENCES


