# Legendárium Navigator:
## Software System for Supporting Transylvanian Tourism

Attila Barna Máté*, Roland Nagy*, Zsolt Szécsi**, Dénes-Bálint Kelemen-Fehér**, Károly Simon***
* Babeş Bolyai University, Cluj-Napoca, Romania
** Codespring LLC, Cluj-Napoca, Romania
*** Codespring LLC, Babeş Bolyai University, Cluj-Napoca, Romania
mateattilabarna@gmail.com
rolinagy94@gmail.com
szecsi.zsolt@codespring.ro
kelemen.balint@codespring.ro
simon.karoly@codespring.ro

***Abstract*— The purpose of the presented software is to help tourists to find their way to legendary locations in Transylvania.**

**The project is composed by a central server, an Android client application and a web client application. The users of the Android application have the possibility to read about the legends, listen to audio books and watch videos. The application provides navigation to legend locations, and a notification system is also integrated for indicating these locations. The required data is uploaded to the server via the web client application.**

## I. INTRODUCTION

Transylvania is frequently mentioned as being "The Land of Legends", with a lot of tales and legendary places. The "Székelyföldi Legendárium" is a project initiated in Szeklerland, aiming to collect local legends and fairy tales (156 so far). Within the framework of the project, a book was published, together with a collection of coloring books, puzzles and other toys for children. Recently, a pilot episode for a 3D animation movie has also been released.

A new initiative is the development of a mobile application for helping tourists to find the geographic location of these legends. The application also provides media content related to these tales and information about upcoming events in the area.

With the help of this mobile application, tourists can read about the legends, browse through pictures and listen to audio books. A map view provides help in finding legendary locations, and navigation support is also integrated. The tourist receives a notification if there is a legendary location nearby. A news feed is also integrated, providing information about ongoing events near these locations. Furthermore, there is a possibility for downloading media content related to legends and in this way the application can be used in offline mode, too. This is a crucial functionality due to the possible lack of mobile signal in some locations.

To provide these functionalities, the required data is uploaded to a central server and it can be managed by Legendárium employees using a web-based user interface.

## II. THE LEGENDÁRIUM NAVIGATOR PROJECT

### A. Requirements

The server is responsible for the business logic and provides a RESTful API for accessing services. Its main parts and functionalities:

- data access layer;
- data importer module (for importing data exported from another server);
- business logic layer;
- user authentication and authorization;
- RESTful API.

The web client provides the following functionalities:

- authenticating Legendárium employees;
- creating, updating and removing legends;
- uploading and managing legend-related media content;
- uploading and managing data for the news feed module;
- importing legends and resolving data conflicts during the import process.

Users having administrator privileges have additional functionalities:

- registering Legendárium employees;
- testing API requests with Swagger;
- visualizing system-related statistics;
- executing health checks on different modules (e-mail, database).

The functionalities provided by the Android client application:

- user registration and authentication;
- viewing/listening and downloading media content related to legends;
- removing locally stored legends;
- displaying legendary locations on a map;
- providing navigation support for the users;
- displaying notifications when a legendary location is nearby;

- trip planning (with legendary locations on the route);
- news feed.

### B. Architecture

The system has three main components: a server, an Android client application, and a web client application. The server can be divided into two parts: the backend and the RESTful API. There is also a common module containing Data Transfer Objects (DTOs) used for communication between the subsystems.

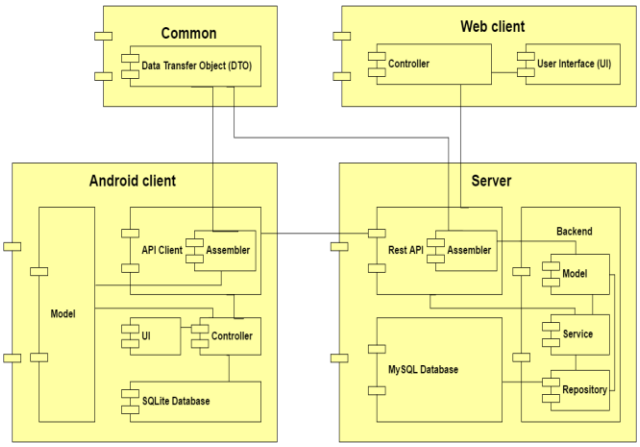


Figure 1. System Architecture

The main entities of the system are included into the Model package. A MySQL relational database is used to preserve these entities.

The Service and the Repository packages are also included in the backend module. The business logic is implemented within the service layer. The required data is acquired from the repository layer. The results are published through the RESTful API.

The API component is responsible for the communication between the subsystems. The API guarantees that the client gets a correct response for each request. DTOs are used for data transfer while JSON is used as a communication format. Each component has specific assemblers for transforming the DTOs into their own models and vice versa.

The Android client uses a SQLite database to locally store data required for the caching mechanism. Both client applications are developed based on the MVC design pattern [1].

## III. TECHNOLOGIES

In the first development phase, skeleton projects were generated for the server and web modules, using the JHipster Yeoman generator. On server side Spring frameworks are used, the web client is based on JavaScript technologies.

### A. The Spring Framework

The business logic of the application is implemented using Spring Beans. These components are managed within the Inversion of Control (IoC) container provided by the Spring [2][3] framework, and the Dependency Injection pattern is also realized using this technology.

Besides the Spring Core framework other server side Spring technologies are also used: Spring Boot, Spring Security, Spring Data JPA and Spring MVC REST.

The project is configured using Spring Boot and an embedded web server is used as runtime environment for the web components.

The authentication mechanism is implemented using the Spring Security framework, based on the OAuth2 standard specification. After a successful login operation, the client receives an access token and a refresh token from the server. The access token is used to authenticate the user, being transferred with each http request. Since the token has a limited life span, the refresh token is used to obtain a new access token after the old one has expired.

The Spring Data JPA framework serves as an abstraction layer over the Java Persistence API (JPA), and the Hibernate Object-Relational Mapping (ORM) framework is used as JPA implementation. Based on this approach, only the repository interfaces are created using the proper naming conventions, the implementations are generated in the background by the Spring Data framework. Moreover, complex queries can be also created within the interfaces using special annotations with JPQL queries.

The communication with the server is based on the REST architecture, which is realized using RESTful web services. The Spring MVC REST framework simplifies the creation of these RESTful web services. The REST resources are Spring components with special annotations. These annotations indicate the type of requests that will be accepted, the type of responses that will be generated. In addition, the endpoint mapping is also specified by these metadata.

The data between the server and the client is sent over the network in JSON format. The Jackson framework is used as JAX-B implementation for serialization and deserialization.

### B. Liquibase

During the development process sometimes the domain model needs to be updated and the database schema is affected by these updates. To manage these changes, the Liquibase framework is used. The change sets are stored in XML format. A special *DataBaseChangeLog* table is created and at each run it is checked if any changes have been made to a table. If a new change set is present, then the database schema will be modified accordingly.

### C. JUnit and Mockito

To ensure the software's quality unit tests were created using the JUnit framework. To perform unit tests, the components have to be separated from each other. If a unit has dependencies, the isolation can be made by using mocking frameworks. Mockito is a Java-based framework having support for mock objects. The original objects can be replaced by these mocks and, in this way, the tests are not affected by the external dependencies.

### D. AngularJS

The web client uses AngularJS [4] as MVC framework. This is an open-source MVW (Model-View-Whatever) framework written in JavaScript. Extra attributes are added into the HTML tags which are interpreted by the framework as directives when the HTML page is loaded.

These directives can be used to bind business models to HTML pages. The models are stored in standard JavaScript variables.

There are four main views within the web application: *footer*, *navbar*, *sidebar* and the *content*. These views have controllers attached to them. The REST resources are accessed by these controllers using specific *factories*.

### E. Bootstrap

The user interface of the web application is created using the Bootstrap open-source HTML, CSS and JavaScript framework. The technology provides support for responsive web design, the pages are loaded dynamically corresponding to the resolution of the user's device.

### F. Retrofit

Retrofit is a REST client framework for Android [5] applications. It uses annotations for sending HTTP requests to the server, and also for data serialization and deserialization (Gson annotations). In the case of the presented mobile application, the data is received in JSON format and it is deserialized in DTOs. These DTOs are converted into model objects using *Assemblers*.

### G. Butterknife

The Butterknife framework is used to bind variables and methods to views. By using annotations, it provides support for dependency injection, generating code which can be reused in many places. For example, omitting the *findViewById* method calls, using the *@Bind* annotation. The listener inner classes can also be omitted, using the *@OnClick* annotation on the method instead.

### H. OrmLite

OrmLite is a lightweight framework for storing Java objects to SQL databases using annotations. The Android application can be used in offline mode. The caching mechanism required to provide this functionality is implemented using this ORM framework.

### I. IcePick

For exchanging data between Android *Activities* bundles can be used. IcePick is a library that resolves code snippets, which appear multiple times when saving a bundle or restoring one. By using IcePick annotations, these redundant code fragments can be eliminated.

The Legendárium Navigator mobile application uses this mechanism to support landscape and portrait mode. It saves the class attributes when the application layout switches between portrait and landscape mode, and reloads these attributes when the rotation is complete.

### J. Picasso

Picasso is an open-source library for downloading and caching images. Some of the main functionalities provided by Picasso and used within the Legendárium Navigator mobile application:

- downloading images;
- resizing images;
- using placeholders while the download process is in progress;
- managing different data sources: files, assets, content providers, resources;
- asynchronous download.

### K. Google Maps

Both client applications use the Google Maps API for viewing legends on a map, based on their geographic location.

One of its main advantages is that it can be customized, besides its basic functionalities (scroll, zoom, etc.) – it is possible to draw on top of the map layer (markers, extra information displayed together with the markers or any other objects).

## IV. Development Tools and Methods

During the development process Scrum methodologies were used, providing an agile software development approach.

Mercurial was used as distributed version control system, SourceTree provided a graphical user interface for source control management and the central repository was managed using RhodeCode.

In the first stage of the development a skeleton project was generated for the server module using JHipster [6].

As build and dependency management system Gradle [7] was used, so the configuration of the project is Groovy-based. In the case of the web module, the build procedure was also supported by Yeoman and Grunt.

For always keeping the code in a correct state the Continuous Integration (CI) development methodology was used, supported by Jenkins. Jenkins was also linked with the SonarQube static code analyzer platform. In addition, it published the stable builds to the test servers and to the mobile beta testing platforms automatically.

Android Studio and IntelliJ IDEA were used as IDEs, together with further development tools (e.g. for database schema management, UML modelling, creating UI mockups, etc.).

## V. Using the Legendárium Navigator

### A. Using the Android client application

After starting the application, a login screen appears. *Remember me* and *Login automatically* options can also be selected on this screen, and there is registration possibility for new users. If the registration was successful, the server sends a confirmation e-mail to the user, containing a confirmation link.

After login, the side menu appears and the main views can be accessed: *News Feed*, *Legends*, *Map, Settings, Logout, About*. A background service is also launched, which alerts the user near legendary locations.

The *News Feed* view displays news organized in a list, with their title, a few rows of their description, and an attached image. A view with a more detailed description is available for each item.

*Legends* are also displayed in a list, each legend on a different card, with its name, region, a cover image, and a book id. The book id associates the legend with its printed version published in the Legendárium book.

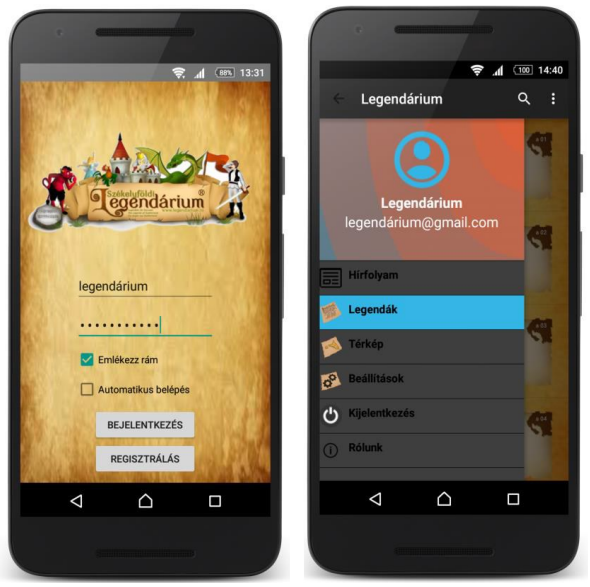


Figure 2. Login screen and side menu

A search feature is also implemented; a dropdown text box appears in search mode. The user has the possibility of searching for legends by their name or region.

On the detailed view of a legend four tabs are displayed: *Description, Map, Photos* and *Media*. On the *Description* tab, the full description and a large cover photo can be seen. On the *Map* tab, the legend's location is marked on a map and driving directions can be requested to this location. The users can listen to audio books associated with the legend using the *Media* tab. The *Photos* tab contains pictures related to the selected legend.

By selecting the *Maps* view from the side menu, a map appears where legend locations are marked. By clicking on a marker, a pop up view appears with the description of the selected legend, and three buttons. These buttons can be used for downloading the legend, and for refreshing or removing the locally stored data. From this view, the user can also be redirected to the detailed view of the legend.

The users can change their personal data and some configuration parameters using the Settings view.


Figure 3. News feed and legend list

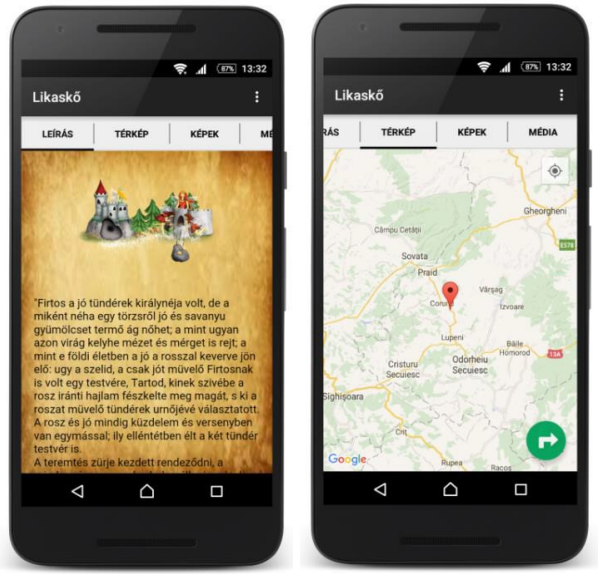


Figure 4. Description and Map

## B. *Using the web application*

After a successful login, the members of the Legendárium team have the possibility of creating, editing and removing legends and related media contents.

There is also a possibility for importing multiple legends at once from a JSON file. There is a special view for resolving data conflicts, which appear during this import process (see at: Figure 6).

The system administrator has some extra functionalities: user management, views with server related information (statistics, configuration, logging, API documentation), possibility of sending API requests via SwaggerUI.

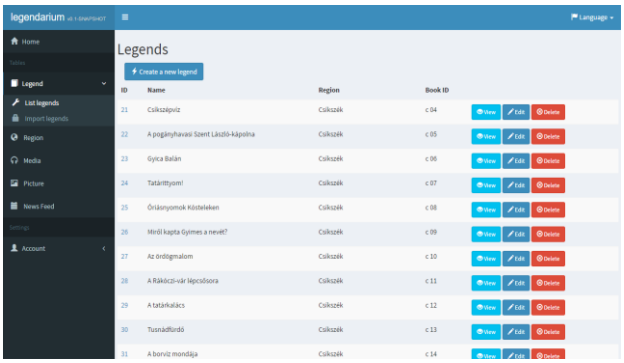


Figure 5. List of legends

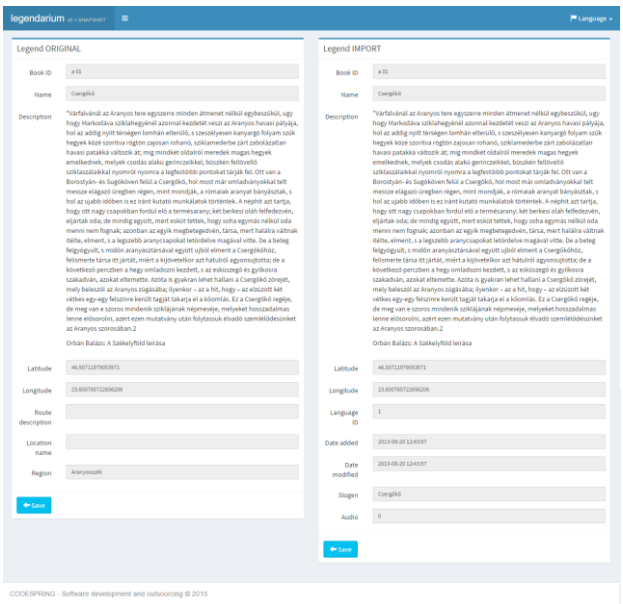


Figure 6. Legend conflict handling

## VI. CONCLUSIONS AND FURTHER DEVELOPMENT

In its current state the Legendárium Navigator system is an operational prototype. After some minor development tasks and a beta testing phase, it can be published and improved based on user feedbacks.

Some further development possibilities:

- Connecting the application with social networks (e.g. sharing events, locations)
- Route planning for trips (the functionality is under development)
- Gamification: after visiting a legendary location the user can be rewarded
- Displaying hotels, restaurants and other POIs near the legendary locations

## REFERENCES

[1] Martin Fowler, Patterns of Enterprise Application Architecture, 1st ed., Addison-Wesley Professional, 2012.

[2] Rod Johnson, Juergen Hoeller and co., (2004-2012), Spring Framework Reference Documentation, [Online], Available: http://spring.io/docs

[3] Chris Schaefer, Clarence Ho, Rob Harrop, Pro Spring, 4th edition, Apress, 2014.

[4] Official AngularJS documentation, [Online], Available: https://docs.angularjs.org/api

[5] Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura, Programming Android, 2nd ed., Sebastopol, California: O'Reilly Media, September 2012.

[6] Official JHipster documentation, [Online], Available: http://jhipster.github.io/v2-documentation/

[7] Official Gradle documentation, [Online], Available: https://docs.gradle.org/current/javadoc/