GeoQuesting: Mobile Adventure Game and Web-Based Game Editor

Beáta Brassai, Boglárka Varga, Károly Simon, Tamás Török-Vistai
Codespring LLC, Babeş-Bolyai University
brassaibeata15@gmail.com
vargaboglarka91@gmail.com
simon.karoly@codespring.ro
torok.tamas@codespring.ro

***Abstract*—The GeoQuesting project is presented, including the functional overview of the components and some details regarding their implementation. Development tools, patterns and technologies are also described. The project includes a mobile game application and a web application for game creation.**

The main component of the project is an adventure game running on mobile devices. While playing the game the player has to solve quests and to find checkpoints in different locations. A web-based user interface is provided for editing, managing and publishing games easily. In this way various quests can be created using different types of conditions (question answering, finding checkpoints based on GPS position etc.).

GeoQuesting can be used in different domains: such as education (e.g.: a game showcasing historical places in a city), tourism (e.g.: a game for touring landmarks in a region), marketing, entertainment industry etc.

***Keywords*— Adventure game, GPS, NFC, Client-Server Architecture, RESTFul Web Services**

I. Introduction

GeoQuesting is a quest-based adventure game running on mobile devices. Players have to complete quests by fulfilling certain conditions in order to progress within the game. The games can be personalised for a wide variety of topics and their structure can also be customised. Depending on how the tasks are formulated and the checkpoints composed, the game can be applied as a tourist guide, it can be used in education or it can be simple entertaining. For example, a use case can be a historical game, during which the player can visit an itinerary of monuments in a town, can be guided to important buildings, and at certain locations he or she can learn information about famous historical figures. The application can lead the player to a location with the help of directions. When the player arrives at the correct spot the game can ask the player to turn at a given compass heading, or it can ask some questions which the player has to answer correctly in order to advance to the next condition. The application also supports conditions that require reading NFC badges. These are placed at given locations by the game creators. If a badge is successfully read by the application, it can be validated that the player indeed has arrived at the correct place.

The project provides a web user interface for the game creation and game management. The users can create new or edit existing games. They can add quests to a game by combining different types of checkpoints (GPS, NFC), orientation conditions and adding questions to be answered. These quest conditions can be freely combined by the editors in order to achieve the desired game. After finishing the setup of the game the editor is able to publish the game. Published games are visible and available for playing to all GeoQuesting users.

II. Development methods and tools

The development process of GeoQuesting is based on agile principles; the team used SCRUM methodology [1] during the implementation. The project uses Mercurial distributed version control system to manage and track changes of the source code. The central mercurial repository was handled by the RhodeCode repository manager. The build and dependency management system of the GeoQuesting project is provided by the Apache Maven software project management and comprehension tool. According to continuous integration principles [2] the project is required to be kept in a compiling and runnable state after each commit. This is achieved with Jenkins, an open source continuous integration server. For ensuring the maintainability of the source code SonarQube is being used to manage code quality. SonarQube analysis is also run with Jenkins.

The web-based user interface of the GeoQuesting project is run on the GlassFish JavaEE application server.

III. Technologies

GeoQuesting is developed using Java based technologies. The server side is developed with the JaveEE platform [3]. JaveEE provides components for helping developers build multi layered component based distributed enterprise applications. The server side business logic is implemented via EJB (Enterprise Java Beans) [4] components; the dependencies between these components are managed using the DI (Dependency Injection) pattern implemented by the CDI (Context and Dependency Injection) framework.

The data access layer is based on JPA (Java Persistence API), which provides an object-relational mapping of the web application domain classes to relational database tables. GeoQuesting is using EclipseLink JPA reference implementation.

The communication between the client application and the server is based on REST (Representational State Transfer) API using the HTTP protocol for data transport. The REST API is implemented based on the JAX-RS (Java API for RESTFul Web Services) standard using the Jersey reference implementation. For serializing the data sent to the client application the DTO (Data Transfer Object) pattern is used [5]. The JSON (JavaScript Object Notation) data format is used as the wire format when transferring data over HTTP in order to achieve cross platform and language independent communication. Documentation and testing of the API endpoints is carried out using Swagger framework.

The web user interface is implemented with the help of Vaadin Open Source UI toolkit framework [6]. Third party Vaadin “add-ons” are used to provide features such as the Leaflet based map view and support for CDI functionality in Vaadin UI components.

The mobile game client is implemented as an Android application with the help of the Android SDK, which provides development tools, libraries and documentation.

For monitoring and tracking the flow of the application the Slf4j logging abstraction framework is used backed by the Log4j as the concrete logging implementation.

The Google Guava library provided EventBus class is used for publish-subscribe based communication between the components of the application. This pattern keeps the communication simple and the components loosely coupled easing the maintainability of the source code.

In order to ensure the correctness and validity of the persisted data JSR-303 specification based annotations are used on the domain model classes to define validity constraints. The Hibernate Validator framework is used to evaluate these constraints before saving the data into the MySQL Database.

In order to increase the productivity while developing the application, the JRebel tool is used for deploying new versions of the complied Java classes under the running container without the need for a restart, thus greatly reducing the deployment time.

On the server side map services are provided by an Open Street Maps layer displayed in Leaflet. On the mobile client the Google Maps APIs are used for navigation and routeing.

The GeoQuesting project uses the Gravatar service to display the profile images of users. The profile pictures are looked up in the Gravatar service by using the user’s email address.

IV. The GeoQuesting project

The following section describes the GeoQuesting project’s requirements, presents the architecture of the application and some implementation details.

A. *Requirements*

Server side

One of the responsibilities of the server is managing the entities in the MySQL database. The Data Access Layer handles the saving, deleting and querying of the entities. The Service layer of the server application builds on the Data Access Layer and contains the GeoQuesting specific Business Logic.

The server side also includes the web user interface, which provides UI for user registration, views for creating, editing and publishing games.

There are provided UI elements for editing the games and the quests that are related to them. For example, when editing a GPS condition for a quest the user is given a map view where he or she can select the desired coordinates by clicking on the map. In order to compensate the inaccuracies of the GPS position provided by the mobile device, the user is able to specify a margin of error, within which the GPS position is accepted. When creating a question based condition the user is able to specify multiple correct answers for that particular question. This is important since for some questions the players can provide multiple correct answers. For example, if the answer is a numerical value, it can be written out using letters or using digits. For easier game management the users are able to sort and filter the games available in the system.

The server also provides a REST API for communicating with 3rd parties such as the Android based game client application.

Client side

The gameplay for the created games on the server side takes place on the mobile game client. The client uses the server provided REST API calls for retrieving the games, but also to upload the progress of the games.

The player on the mobile client can browse the list of the published games, view the details about a given game and also select one for play.

After starting the game the user is presented with different quests that the user needs to complete in order to finish the game. A quest can be made up from one or more conditions. The application presents the conditions in order, one at a time for completing.

While playing the game the user will encounter different views for different types of conditions. For example: a location based condition in order to be completed requires that the user arrives at the given GPS position. When solving a location based condition the user is presented with a map and a recommended route for reaching the correct position. When confronted with a direction based condition the user is shown a compass to help him in the orientation and to turn to the appropriate heading.

B. *Architecture*

Figure 1 describes the system's components and the relationship between them.

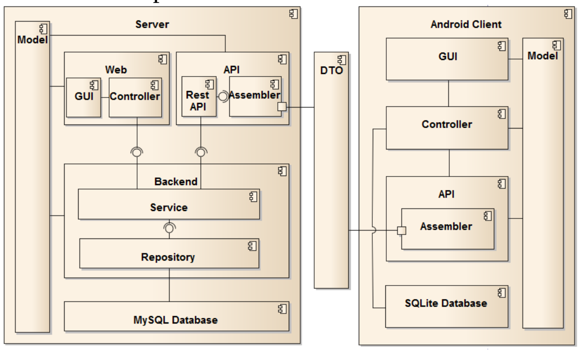


Figure 1. GeoQuesting's architecture

The appropriate Java classes for the domain model entities of the server application are found in the model package. The backend, Web UI and API components rely on the model entities.

The backend component includes the Data access and service layers. The low level data querying, inserting and deleting is implemented in the repository layer using the JPA Entity Manager. The service layer interfaces provide the application logic to the web user interface components and to the REST API.

The web component provides the interface for the user in order to access the application. The user interface components are built according to the MVC pattern. This way the different concerns are separated among the classes. The view class is responsible for displaying the model data. The controller handles the actions performed on the data and forwards it to the service layer.

A similar structure of the components can be found on the mobile client application. There is also a model package but this contains the client specific model classes.

Largely, the model classes are the simplified versions of their server-side model counterparts. This approach was chosen because the client application only uses a part of the information that is stored on the server side.

The communication between the two main components of the project, the server and the client side is provided by the REST API. The REST API uses DTO objects to transfer the data between the API endpoints. Assembler classes are used to convert the server and client model objects to DTO objects that are serialized into JSON and sent over HTTP.

The mobile client application uses a SQLite database to store user data. The Controller component is responsible for accessing the data; for business logic operations; as well as for handling the data from the device's sensors.

The client side application provides an easy-to-use interface for the visualization of conditions (showing maps, displaying a compass) or tracking the progress of the games (showing progress bar) etc.

C. *Implementation*

Condition evaluation

For evaluating the completeness of the conditions different device sensors are used by the client application.

When evaluating a GPS based condition the application uses the device's GPS signal receiver. The position determined by the sensor is compared with the position found in the quest, if it is within the given margin of error the condition is considered fulfilled.

Orientation conditions are validated by reading the compass heading from the device's magnetometer. If the reading is within the interval specified in the quest, the application accepts that the user indeed has turned to the correct direction with his mobile device.

Because the availability of the GPS signal is limited inside buildings, the position of player can be determined with the help of NFC badges that are placed at certain places and read by the NFC module.

NFC is a communication standard based on short range (less than 10 cm) radio waves. The NFC protocol does not employ security settings at all; connection is established by placing the NFC enabled devices near each other. The lack of setup provides an easy and frictionless gameplay while reading NFC badges during quests.

When solving question-answer type conditions in order to compensate for spelling mistakes or lack of accents use by the users the Jaro-Winkler distance algorithm [7] is used for answer validation. The mobile client computes the distance, represented by a number in the range of 0.0 and 1.0, between the answers provided by the user and the correct answers supplied by the server. The closer the distance number is to 1.0 the more similar is the provided answer. When computing the Jaro-Winkler distance the algorithm takes into consideration the number of matching characters and the number of transposed characters of the two strings and also the length of common prefix at the start of the words. If this value is above 0.85, the game accepts the answer provided by the player as being correct.

Data transfer

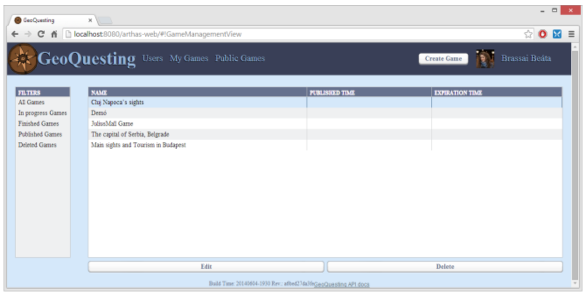
Data exchange between the server and game client side is accomplished by the use of DTO objects. These objects are constructed from the corresponding model objects by the use of Assemblers. The responsibility of the Assembler class is the conversion of domain model objects into DTO objects used by the API and reconversion of the DTO objects received by the API into domain model objects. The assemblers and DTO classes also support the serialization / deserialization of model objects that are organized in a class hierarchy. When for example un-marshalling from JSON a child DTO object the assembler will automatically create the correct child domain model object and it's ancestor objects. This way the API supports the polymorphic conversion from DTO to domain model objects.

D. *Case study*

From user's point of view GeoQusting provides two application interfaces: the web-based game editor interface and the mobile game client application.

For presenting the functionalities of the GeoQuesting project, the following example is presented.

After successful login the user is present with the main of the web application. The user is able to view the list of already existing games in the system alongside with information about them (Figure 2).



NAME	PUBLISHED TIME	EXPIRATION TIME
Cluj Napoca's sights		
Demó		
JuliusMall Game		
The capital of Serbia, Belgrade		
Main sights and Tourism in Budapest		

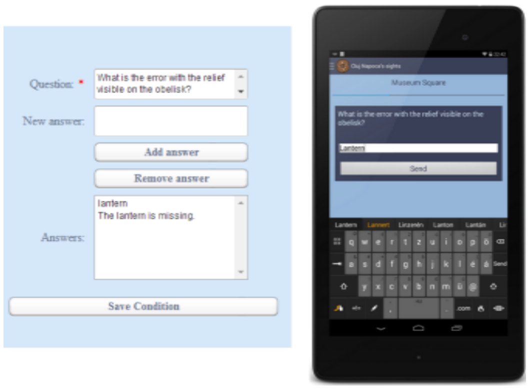
Build Time: 20140604-1930 Rev.: a6bed27da3fe GeoQuesting API docs

Figure 2. Web game editor interface - Public game list

When creating a new game the editor has to specify the name (in the example "Visit Cluj-Napoca's sights") and short description. Optionally the user can provide a region for the created game (e.g. "Cluj-Napoca") this way the users will have the ability to search for games that are playable in a certain region close to them.

After creating the game editors are required to add one or more quests to the game that the users have to complete in order to finish the game.

Quests are created by combining different types of conditions. These can be put together freely as desired by the game creator.


Question: * What is the error with the relief visible on the obelisk?
New answer:
Answers: lantern
The lantern is missing.

Cluj Napoca's sights
Museum Square
What is the error with the relief visible on the obelisk?
Lantern

Figure 3. Question-answer condition

When creating a question-answer condition (Figure 3) the user specifies the question (e.g. "What is the name of the main square of Cluj-Napoca?") and the list of possible answers to the question. The answers for the question can be varied e.g. the Hungarian or Romanian name of the city Cluj-Napoca. In addition to multiple correct answers provided by the editor the mobile client will employ fuzzy text matching based on the Jaro-Winkler distance in order to compensate for some spelling or typo mistakes.

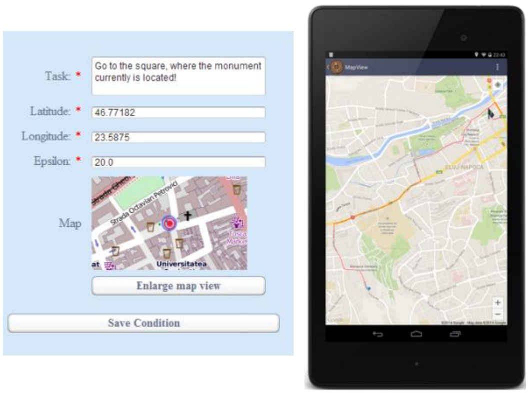

Task: * Go to the square, where the monument currently is located!
Latitude: * 46.77182
Longitude: * 23.5875
Epsilon: * 20.0

Figure 4. GPS condition

If the game requires that the player is at a given location (in our example at the Main Square of the city) the editor can add a GPS location based condition (Figure 4). The editor can specify the position by making a selection with a mouse click on the map. The editor can also specify a radius around the selected position within which the player's position is considered as being at the correct location. When playing the game, the player is provided with a map and a recommended route from the user's current position to the quest objective (location).

If it is important that the player is looking in a specific direction, e.g. toward the statue of King Matthias, the game editor can add a direction based condition. This is done by providing an angle range e.g. from 90° to 120°. During gameplay the player is helped to face the correct direction by displaying a compass based on the devices magnetometer (Figure 2).

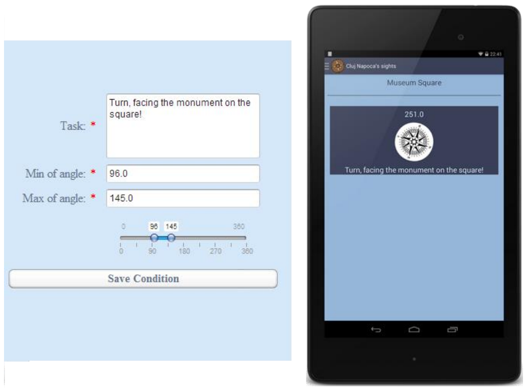

Task: * Turn, facing the monument on the square!
Min of angle: * 96.0
Max of angle: * 145.0

Figure 2. Direction condition

If the position of the player needs to be determined in places where the GPS reception quality is low e.g. inside building, editors can add NFC based conditions to the games. When creating the condition editors enter the unique id of the NFC badge. Then they place these NFC badges at the locations where they want the users to be. When playing the game the players search for the physical location of these badges in the field. When these badges are located by the players they touch their devices to the badge, the device will read the unique id of the badge and will compare it with the server provided one. If it matches with the one provided by the game editor, the game validates the location of the player.

Some conditions can be specified multiple times per quest e.g. question-answer conditions but others only once e.g. GPS conditions. If the game would allow specifying multiple GPS conditions per quest, the quests would be unsolvable since the player cannot be in multiple places in the same time. A game can contain as may quests as the game editors desires.

After finishing the editing of the game, the editor can publish the game making it publicly available to other users for play, optionally giving a validity period.

The players can start games at any time, and can also postpone the completion of the game. Their progress is automatically saved and they can resume the game at a later time and finish it.

V. Conclusions and further development

GeoQuesting is an entertaining game useful in different domains. Games can be easily customized using the web interface. The system is easily extensible, scalable and maintainable.

There are several further development possibilities. New condition types could be introduced, for example places identified by barcodes for devices without NFC reader. Augmented reality contents could be displayed on the camera view.

A gameplay feature could be introduced for players to share virtual objects with each other using NFC.

An internal evaluation (games rating), score (rankings) and reward system could be evolved. Automated filtering possibilities could be provided for the games (e.g. based on user’s location or features supported by the mobile device).

References

[1] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, Prentice Hall, 2006.

[2] P. M. Duvall, S. Matyas, A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, Addison-Wesley, 2007.

[3] A. Gupta, *Java EE 7 Essential*, Sebastopol, California, USA, O'Reilly Media, 2013.

[4] A. E. Rubinger and B. Burke, *Enterprise JavaBeans 3.1*, 5th ed., Sebastopol, California, USA, O'Reilly Media, 2006.

[5] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002.

[6] M. Grönroos, *Book of Vaadin*: 4th ed., Turku, Finland, Vaadin LTD, 2013.

[7] W. E. Winkler, *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage*, Proceedings of the Section on Survey Research Methods, American Statistical Associatio n, pp. 354–359, 1990.