SCORE Contest Project

Project: BTWmaps

(By the way, if you go my advice to you)

Available at http://btw.rasip.fer.hr

Team members:

Nikola Tanković (nikola.tankovic@gmail.com)

Gianluigi Ciambriello (g.ciambriello@gmail.com)

Sonja Miličić (sonja.milicic@gmail.com)

Danijel Zović (dzovic@gmail.com)

Savino Ordine (savino.ordine@gmail.com)

Zafar Amhad Bhatti (bhatti.zafar@gmail.com)

FER

MÄLARDALEN UNIVERSITY
SWEDEN

Executive Summary

The goal of *BTW Maps* project is to provide extra features to map and route generating software like *Yahoo! Maps*, *Google Maps*, *Via Michelin* and other. These programs, in addition of providing maps, also calculate the optimal routes for travel, but they lack any other information about these places. Therefore we want to extend these maps with user-supplied advices. We believe that the individuals who live and travel in a city can be a fountain of useful information, so they can use their knowledge to help others.

As the project idea opens a variety of everyday use and from the very beginning it is easy to predict that *BTW Maps* will very soon grow and spread its use on mobile devices, our main goal was to design the architecture so that it is easily extensible as well as friendly for these new technologies. This kind of service-based architecture also deals with our main problem, which is inability to predict all of the use scenarios because *BTW Maps* is targeted at all audiences. So our architecture is prepared to enhance functionality as the project starts to spread among users to fill their needs.

This project is developed inside the Distributed Software development course provided by Mälardalen University – Vasteras and Faculty of Electrical Engineering and Computing – Zagreb. The aim of the course is to provide knowledge on how to deal with problems related to distance and different cultures in developing software in a distributed environment. The team of the project is composed of three students from Vasteras and three from Zagreb. The main constraint in the project was reduced amount of time since semester in Vasteras starts something later then in Zagreb, so instead of full five months, we had only three months available. This was quite challenging in spite of the fact that we were assumed to work with double efforts that for courses running 5 months. That was only a stronger motivation for us to start work immediately and to plan our activities carefully.

There were certain issues with gathering requirements for this project, as we were all rather new in public relations. We first did some market research and queries with different people groups and communities. Each of our investigation attempts produced a set of new requirements, and with establishing contact with these groups, mainly students and people with disabilities, new requirements would come in regularly. So given this reduced time schedule, we had to work fast and efficient, and to make it we needed to do much work in parallel, and in several iterations. Careful testing of each new feature and regular integrations each week was inevitable to review all the work and to fill the documents required by our course supervisors.

BTW Maps was implemented using modern technologies (*PHP, AJAX*, *JSON* …) and it is mostly Object Oriented which makes it easy to maintain and extend. Whole architecture is divided into modules each with its unique function. Server modules are coded in *PHP*, client modules in *Javascript* and data is exchanged in *JSON* format.

Our work result, *BTW Maps*, is a system that it is easy to use: to see all of the community entered advice on a user's planning route or for a general location. With registration a user gains further abilities like customizing your profile, selecting your desired advices and properties to display automatically on map, as well as your home location for quick route searching. You also gain ability to enter your own advices, and if you prove yourself worthy you can also be selected for moderator or administrator by existing set of system administrators. Then you have the ability to enter new advice categories and spread the system even more. The quality of our system has already been recognized because in 5 days of its existence, we already have more than 30 users registered and more than 70 advices given.

Now that *BTW Maps* lives, we are very pleased with the outcome as well as our stakeholders. We think that the main reason for that was good team atmosphere and coordination. Stakeholders were especially pleased with our intuitive interface and design produced for this project by our designer, who also made sure that the whole system is user-friendly for best user experience.

1 Document overview

This paper starts with the requirements we are supposed to fulfill in *Section 2* and the problems we stated in *Section 3* followed with *Section 4* in which we define non functional requirements. Our management plan is presented in *Section 5.* Then we state how we planned our work in *Section 6* and show some details of our project architecture in *Section 7*. Section 8 discusses the implementation methods and *Section 9* the testing. *Section 10* contains information about the development process used and in *Section 11* we bring the outcomes. Finally we state what we have learned and summarize the whole paper.

2 Requirements and Problem Statement

The ordinary maps can show routes and some directions online but think about cases when users plan a trip to an unfamiliar city and require more details! Once they have a route which they think is reasonable, they can ask what others have to say about that route and can get some specific details which they need. We aim to make a route-planning system that allows community input, thus providing extended GIS information. That kind of input will provide a better way of travel and benefit users making them comfortable to travel across new destinations by having the advices and warnings easily available to them.

Our system must provide solutions for the below stated requirements efficiently and easily:

1. Provide a way to insert Departure and Arrival destination

2. Provide users to choose the type of travel offered by the route-finding tool as

    - Driving,
    - Walking,
    - Public transportation.

3. The system should be able to store and retrieve advices from the users which include the information about the places like dangerous areas, rush-hours, construction hazards, parking places, ATMs, underpasses, and flyovers etc.
4. Registered users have to be able to write advices in multimedia format, including text, pictures, video, and audio.
5. Users can flag advices as inappropriate to make the system more reliable and authentic.
6. Provide the facility for the disabled and old people to travel through the difficult areas by using advices have certain properties like “wheelchair accessible”, “well lit”, “blind people friendly”, etc.
7. Register the users and seek through their personal profile so that advices that fill their needs automatically gets displayed.
8. Provide users with secure and reliable system as much as possible, so that any fake or wrong advice doesn’t get place on map and cause misguidance.

3 Requirements Specification

The primary success of the software system is the degree to which it fulfills the needs of its stakeholders and users. As the requirements are not targeted to a specific group of people because everyone can use this system for easiness and comfort in travel, we had to generalize our system. We had to ensure that even if the requirements evolve through the passage of time, we will be able to handle them and make an efficient solution. To make this system more general we gathered the requirements from people located in different places so that

we can make requirements more general. Luckily our team is distributed so we gathered information from different countries stated later below.

3.1 Requirement Gathering Methods

The requirement gathering takes out sufficient time to receive all possible requirements from the people who wish to use this system and are ready to help others by giving their advices. We introduced some questionnaires with some specific questions and also the opinions and suggestions from diversified people to know as much as we can to start our project in a good way. The main methods adopted to gather the user requirements were:

- Questionnaires
- Personal Interviews
- Online group discussions
- Mails

The questionnaire included twelve quick questions, for example:

- ✓ How important to you is to show tips about dangerous areas of the city during the day or night?
- ✓ How important to you is to show information about the shops?
- ✓ Do you need to keep the details about flyovers, underpasses, traffic signals and timings for a route?
- ✓ What advice do you need most and how do you need to use the system?
- ✓ Suggestions, advices, opinions

3.2 Statistics

The people who gave us their input and wish lists were from different continents like Asia (Lahore, Rabwah, and Karachi in Pakistan), Europe (London, Frankfurt, Zagreb, Vasteras, Rome, Stockholm, Helsinki and Uppsala), and America.

Concrete numbers:

- 40 people answered questionnaire
- 25 personal interviews
- 100 new ideas through e-mail
- Advices gathered from over 10 large cities and over 6 countries

The questionnaire was sent to people located different parts of the world. Personal interviews were carried out at the University of Mälardalen in Vasteras and Zagreb University in Zagreb. The interviews were also conducted within the cities Vasteras, Lahore, London, Rome and Zagreb. The group discussions and mails were also sent to more than 100 people who responded with some good ideas and opinions e.g. ATM points, Hotels, Parking Places, Historical Points, night transport, restaurants etc. This process was a milestone to understand the requirement and the elicitation related to them.

The requirement gathering phase included the personal interviews with the general public but was conducted especially to small group of disabled people who have the low-vision problem or are wheelchair users. It is also kept in mind that the people with other problems can also use this system, so the system should be very prone to change according to their needs. The questionnaires, mails and online group discussions also had some pictorial and graphical images to make easy, useful and attractive interfaces which can support the changes without having effect on the other requirements and development processes.

3.3 Communication

Most of the communication with the intended users was done through the internet, chatting and groups due to the distributed environment within the team members as well as the view to make the generic software which could handle the needs and requirements of the people living anywhere in the world. The process also contained a prototype and a graphical representation of the system which were obtained from the SCORE requirement document. The prototype and graphical representation also helped out in validating and analyzing the requirement so that we make sure that the system performs the actions what indeed the user wants.

3.4 Advice Collection

The method adopted to understand how the advices should be stored and retrieved from the system was *Cognitive Psychology* which provides how the people find the difficulty in describing their needs. The use case 'Add Advice' illustrates this requirement in the *Figure 1*. We took the opinion from the users in their favorite languages and with some examples to save and show the advices.

The web techniques which are supported by the different technologies are available to speed up the data from database and can display the advices without any hindrance. Even when the user moves up or down the map, the new advices have to be carried out with sufficient good amount of time to be shown to the user. What makes this possible is use of *AJAX* [14] methods to retrieve and store information to/from server.

3.5 Dynamic Behavior for Requirements

For the requirement elicitation we observed our users very consciously and we wanted to enable them to use our system during development and see what more do they require of us to add. Prototyping technique can be used to deal with the uncertainty about the requirements and early feedback from the users.

The context and the ground of the requirements was taken care of very keenly because the type of advices and the way to handle them may evolve during the development so the layering architecture can support it with great deal because then we don't need to change much things but simply the things related to a specific tier. Also it will not be expensive and hard to change even if more advices related to different groups of people come into the way. The presentation and interface layer is very easy to change and modify so that the disabled people can use and customize it through their easiness and system can bear the changes in every stance of time to cater these things.

For groundwork and associated risks to the project requirements and the dynamic behavior of the requirements we have chosen *N-Tier Web development platform*. It makes it very easy to change and incur any new thing in the system just by doing our work in independent fashion.

3.6 Use cases

In *Figure 1* you can see all the basic features we were supposed to make. We have 4 classes of actors:

- Anonymous Users
- Registered Users
- Moderators
- Administrators

Anonymous users can find routes and check advices, but if they want to add new advices they need to register. Registered users can also "Flag" advices to note the administrators something is wrong or inappropriate. Simple user can be promoted by an Administrator to Moderator so he/she can help to manage flags and users. Finally administrators can do all the things available to registered users and also delete users from the database and promote/demote a user.

Here we put two examples of use cases:

- Route Finding

  To provide the arrival and departure destination by the user, *Google Maps* [1] seems to be the most suitable way. They have been used and tested by the people around the world and they are easy to use and have some faster ways to extract out the paths and routes. The use case 'Find routes' was aimed to fulfill this requirement. By default BTWmaps can show users their own home address or profile address as departure or destination point so that user can feel easiness and know the way to input the route.

- Area Selection for Advice

  The user can make a marker at a point to represent his advice but if he targets a big area on the map, then there can be a polygon or a line to save the advice for that specific place. User can drag the mouse to select an area on the map and has controls for zooming also. The coordinates of the lines and the polygons are saved to database for later representation in advice giving.

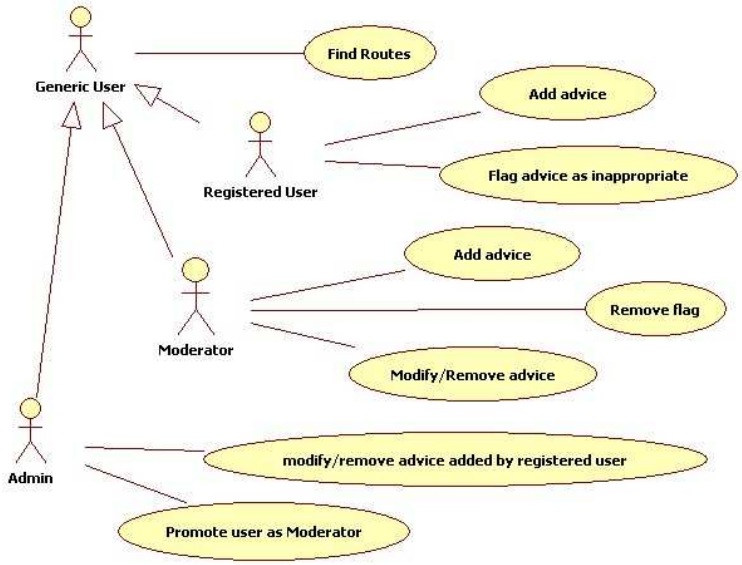

Figure 1, Use case diagram

4 Non functional requirements

4.1 Usability and Easiness

To enable user more comfort and not to mess up with all the advices on the map at a time, it is advised to show the user just those advices which are relative to the user's profile by default and then also made it possible to change the view of the advices by zooming in and zooming out and by customized filters to show the advices as the user wishes. So in each moment user can filter advices simply by clicking on them on menu and advices that concern users profile are automatically checked.

It is also necessary to note that each icon showing advice has tags related to it to enable work of screen readers for disabled people. We used techniques provided to us by *HTML* [2] language to enable work of screen readers and similar helping tools.

Some of the techniques used, as in *WCAG2.0* [3]:

- Combining adjacent image and text links for the same resource
- Providing text alternatives for the area elements of image maps
- Using *ALT* attributes on *IMG* elements

4.2 Reliability and Authenticity

To make the advices authentic and reliable the administrator and flag system is used so that the users can comment on an advice and also the administrator can filter out the unreliable advices from the system. It can be done with the flagging on advices and the administrative section of the system. Increasing number of advices will make the administrators burden more and more, so we will have a mechanism to promote the good users as an administrator or moderator to give the helping hand to existent administrators. Moderators have only the means to modify all advices but administrator along with that also has the ability to remove users and flagged advices as well. The use case 'Promote user as a Moderator/Administrator' can explain this requirement in the good way. The system should provide a protection of user profile data held in the database. It is very important to gain the confidence of the user and also make his personal information secret. To accomplish this idea we can use a secure and authentic way to store password in *hash* form.

4.3 Scalability

The system should also support the large number of users and their advices because the Google map can almost search all around the world so the user can save advices for any place he looks for. To handle this task there should be flexibility in the system to bear any new database and new technology to support the load of the users. The separate layers will contain a good reason to gauge this requirement in independent fashion.

5 Management plan

5.1 Team introduction

Our team consists of three students from University of Zagreb, Croatia, and three international students from Mälardalen University, Västerås, Sweden. We are all students of computer engineering with no previous knowledge about distributed software development so this was a great chance for us not only to learn something new but also experience full concept of distributed development.

Team members and nationalities are as follows:

Zagreb team:

- Nikola Tanković (TN), Croatia – Project Manager and Zagreb Team Leader
- Sonja Miličić (SM), Croatia
- Danijel Zović, (DZ) Croatia

Västerås team:

- Gianluigi Ciambriello (GC), Italy – Västerås Team Leader
- Savino Ordine (SO), Italy
- Zafar Amhad Bhatti (AZ), Pakistan

5.2 Team expertise

From the total of six members in our team, two already had advanced knowledge of web development, two members had somewhat experience of web page development, and two members had only theoretical knowledge and expertise in other areas like object design and database modeling. All of this had to be taken in consideration when dividing roles among the team.

5.3 Team coordination

On top of all, and common practice among distributed development, each team has chosen itself a team leader which is responsible to coordinate his own team. Instead of conducting communication between each of members, we agreed to pass the most important project communication just between the two of team leaders, and make regular Minutes of Meeting which consisted of all of the topics discussed together with conclusions. Each team leader would then consolidate with its team. Our Project Manager, on the other hand, is responsible for all of the administrative tasks and therefore he must breathe with the project itself.

5.4 Programming management

On the very beginning our team in overall was rather inexperienced in actual web programming as we had only two of six programmers with advanced knowledge in all of the project technologies, so there was a time needed for other to educate themselves into specific programming languages and techniques. During this period our two main programmers put a considerable and focused effort to come with an implementation of system core both on client and server part as this was the most difficult programming task. Second, which we have not predicted, system core was rather difficult to understand for the other two programmers. To overcome this there was an additional need of education period which required active collaboration of main programmers with other programmers. This education was found rather difficult to accomplish in distributed environment as

documentation itself was not always sufficient. The implementation of the system core was done accurately, so we found easy to work with that after we get familiar with it. An advantage was that we focused on a component-based design with clear specifications all that helped our distributed work, saving efforts to put together components.

5.5 Communication

A good communication is vital part of any distributed project, especially ours because of our "agile" development we always needed to be synchronized with our work. There were strict rules on reports of each team member as we all needed to fill a week report regarding our work and future work as well. Once all documents were gathered in the end of the week, we summarized it in one Team week report. Based on that report, initial project plan, and on new requirements as well, we had telephone or video conferences on Mondays regarding work to be done during that week. When work was assigned smaller groups were formed which communicated extensively during development on that week.

Each action and system coding was followed by documentation that is posted on our Google group for others to study and use. Big discussions were also taken through that group and each member posted his own opinions regarding actions to be made. Smaller discussions between two members and big conferences were made over MSN, but we also used Skype voice calls and sometimes video conference (from our universities' conferences room) for some complicated system explanations and education.

5.6 Managing project artifacts

All of the project files are located on our *SVN* server to which each member has access. Every one of us had also our local checkout on which we developed new parts of system. After each unit or feature is developed, the code is sent to main server and merged with others code if necessary. Strict rules on coding style were also posted on group page which was our main document repository.

5.7 Integration and tracking

By the end of each week integration of components developed through that week took place online on our main server (http://btw.rasip.fer.hr). That way we made sure that all parts worked not only on our local environment but in production environment as well. We used this website as our primary demonstration point for our stakeholders to gather their opinions and new ideas, because as system spread, there were more constructive suggestions available to us. Also we used this site to make clear for our supervisors that the project is on track, as they were able in every point of the project development to check project status and implemented features we stated in our reports.

6 Project Plan

As *BTW Maps* are composed by different modules (see *section 6.3*) and developed by different people, we made a project plan in the first week to have a good organization in both timing and tasks. Following our knowledge in software engineering we created milestones for each phase and we divided the project in server side and client side with sub-categories.

6.1 Dividing tasks

As shown in *Figure 2* we had different tasks to work on. Some of them were required to be completed before starting others; others need to be done in parallel. We tried to imagine how the tasks flow was and how long each task would have taken. In the starting phase of the project we made the project plan following the waterfall model, defining all phases and all milestones strictly. Then we realized it was too hard to have this constraint and we decided to make it more adaptable to changes. So, following the *agile process* (see *section 10*) not all the phases were well defined, for example we collected requirements in the early project status, but also later during development. Fortunately we succeeded in following the plan with some small modification on the planning flow (for example we needed to modify the database to improve the quality of the management).

The Gantt diagram we have created before starting the implementation phase is shown below. It has all the phases well defined and it is linear in time. On the right is the actual diagram showing how almost all the phases are extended.

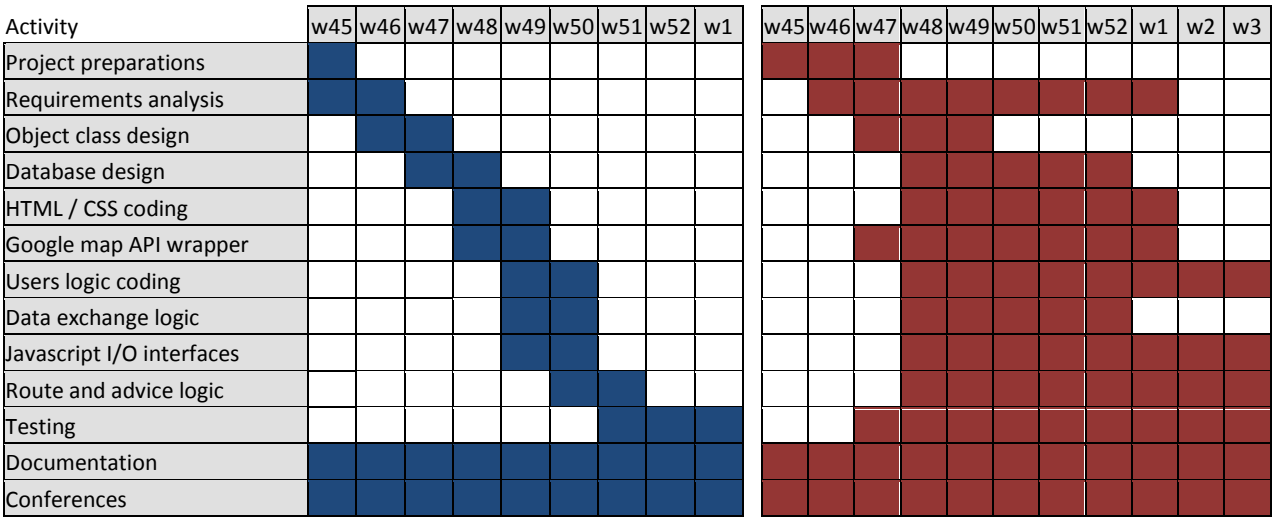

Figure 2, The Gantt diagram created before start of developing on left and actual one on right

6.2 Milestones

By dividing the project in milestones we checked our work and avoided to go out of time. We started studying the requirements to know what the customers want and what is feasible with the technologies available (we had to include external technologies like *Google Maps API* [4]); then we designed models to clarify ideas and control all the parts avoiding problems in the future for parts not considered. After finishing this part we created the database and then started the implementation phase.

We included testing in each phase and left the last week for global testing.

Id	Milestone Description	Responsible Dept./Initials	Finished week			
			Plan	Forecast		Actual
				Week	+/-	
M001	Requirements analysis & definition	TN	w46	w46	0	w01
M002	Object class design	AZ	w47	w47	1	w49
M003	Database design	DZ	w48	w48	2	w52
M004	Map wrapper / HTML/CSS	SM, AZ	w49	w49	3	w01
M005	User server logic, JS interfaces	GC,SO,TN	w50	w50	4	w03
M006	Route and advices	SM, TN	w51	w51	5	w03
M007	Testing and documentation	TN	w1	w1	6	w03

Figure 3, Milestones - See section 7.1 to members match member names with initials.

7 Architectural Design

7.1 Motivation and overview

As it was very hard to come up with a complete list of requirements for this project, as well because our customer was not "real" in terms of buyer who knows what he wants and because our project requirements include only the information of small group of potential users, so we needed to make sure that the architecture will open to future extensions and new use cases.

We found that using real *SOAP* [5] or *REST* [6] standards were rather complicated for our project due to lack of time, so we built a simple representation of *SOAP* based on *JSON* [7] rather than *XML* [8] to exchange data and used existing libraries for client-server communications. The only negative consequence of this approach is absence of some kind of broker like in regular web service, so when implementing we need do carefully document input and output service objects know as messages in regular *SOAP*.

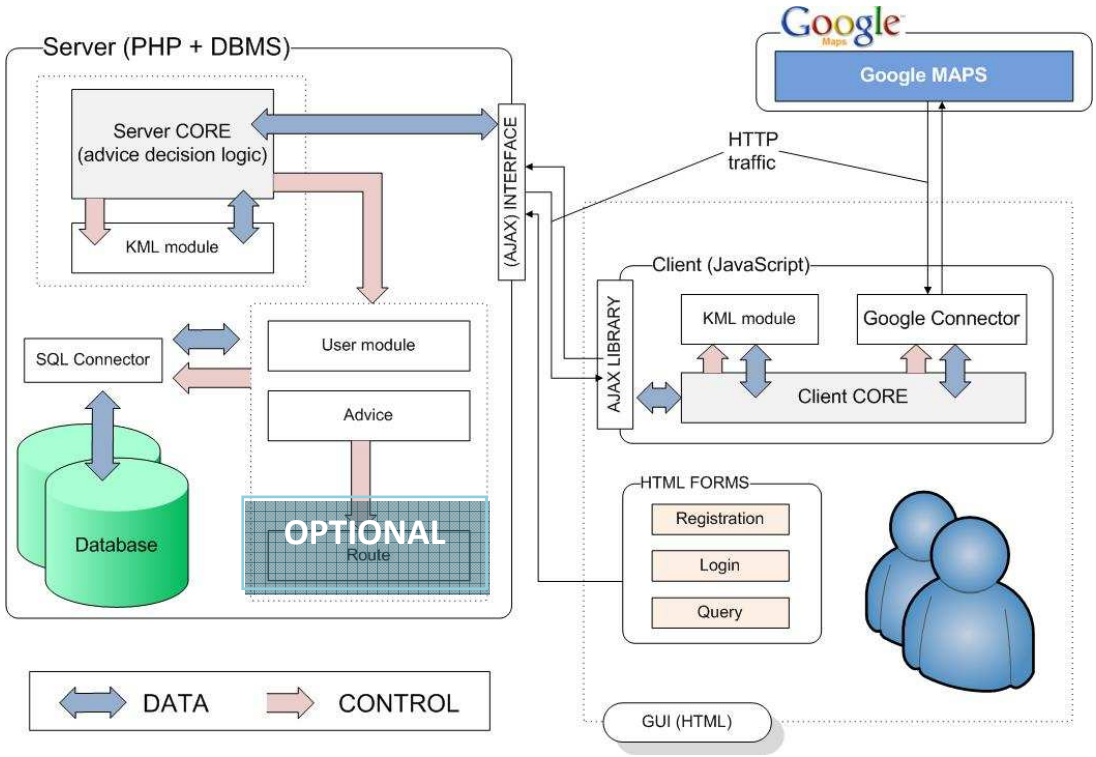

Figure 4, Architectural design overview

As shown in *Figure 4,* whole application is divided into server and client parts that are divided in modules. More on the specific parts is described in next 4 chapters. We used open source *PHP* [9] language for the server part due to years of experience of certain members of our team, so that we minimize learning time as much as possible. Also, our team was competent in *Javascript* [10] coding as well, so we had certain advantages in these technologies.

Technologies used:

- *PHP* object oriented programming
- *AJAX* calls to server
- *JSON* data exchange between server and client
- *Javascript* object oriented programming
- *Javascript* asynchronous programming

We used PHP object code to implements all server modules. AJAX technology was used to make queries to server from client Javascript code, and these queries transferred data in JSON format. Most of the client code is also object oriented and coded in Javascript, and asynchronous programming methods were used to handle events.

7.2 Database

The main purpose of server service is to deal with database information. We of course need database to store all of user information, advice information and their placement on world map. We aimed at good open source reliable database management system (DBMS in further text) with good integrity preservation so were faced with two possible solutions: *MySql* [11] or *PostgreSQL* [12]; again due to good past experiences with PostgreSQL it was our logical choice, but the architecture has a database abstract layer so it can support even MySql (or other) if needed in future releases.

We started with Entity-Relationship database structure (*Figure 5*) for easy future extensions, but the tests showed some drop in performance due to many table joins in example of fetching advices. Testing showed that over 95% of database access is used to fetch advices, and each advice set, or set of advices on certain map area, required for DBMS to perform three table joins. So we needed some kind of simplification without much loss of extensibility and came up with a 2-table Object Relational model for advice storing (Figure 5) which required only one table join to be performed.

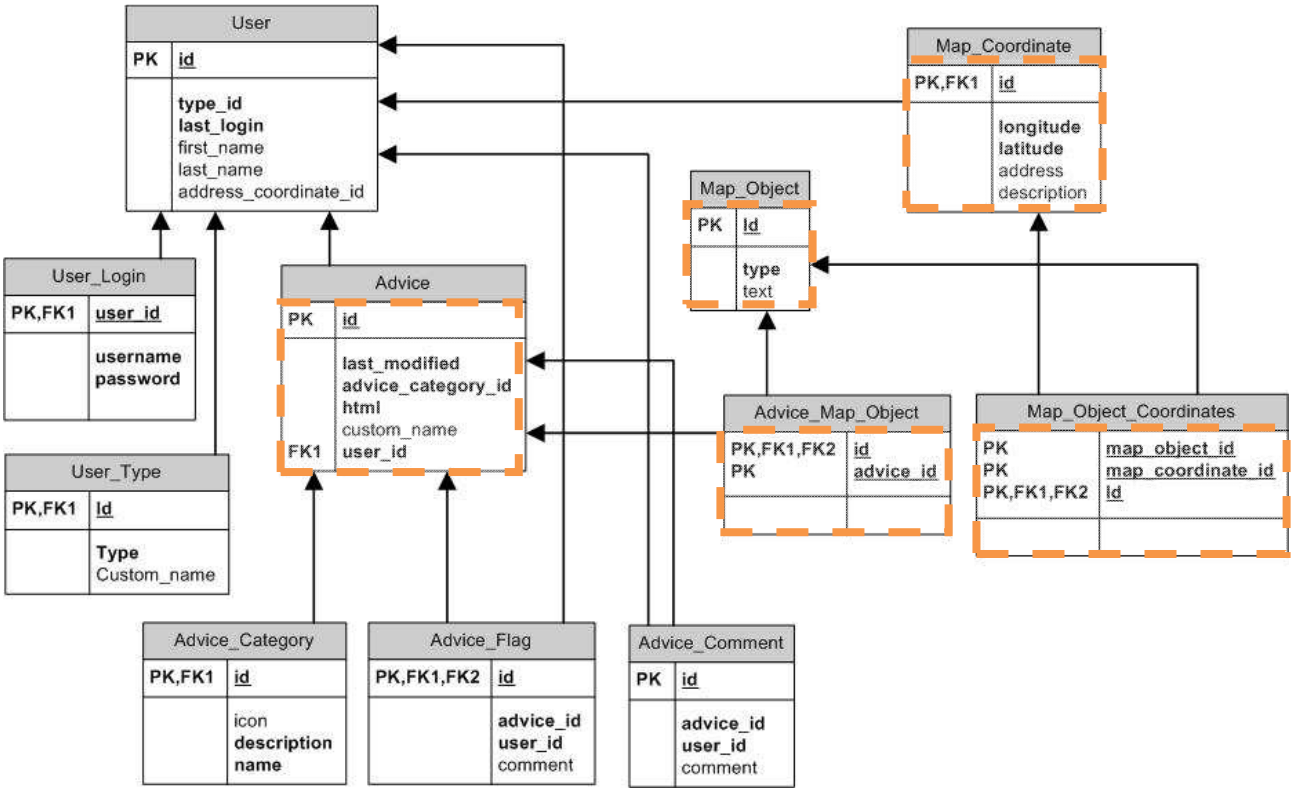

Figure 5, Previous DB model with 5 tables for advice information

7.3 Server side modules

Server side architecture consists of several modules (see *Figure 4*). These are:

- Server core
- Service interface
- User module

- Advice module
- Session module
- SQL module

In *Figure 4,* you can see a module named “Route”; this is an optional module since Google API provides us with routes calculation, but this module leaves open to specify in future releases some custom routes, and for users to provide special routes and shortcuts.

7.3.1 Server core

Server core is where our service mechanism is located. Actually it consists of main server class, and subclasses which implement different ways of client communication types with server class methods. These classes are service interface. Server class methods are actually all of the *BTW Maps* functionality that can be called by any type of client supported. Currently we have service subclasses supporting calls from *HTML* forms for normal user interaction (registration, editing profiles, etc.), *AJAX* calls (advice fetching and storing, flagging, commenting), and internal server calls which are the calls from system itself. This being all of the data flow goes through this main class and then is diverted into help modules as needed.

7.3.2 Service interface

All of the calls to service are handled in the same class and in the same methods, so we don’t differentiate clients. Actually all of the logic never duplicates in any way, which brings to good extensibility and maintainability, so if we for example decide one day to build *iPhone* [15] application that enables user to send photos “on site” and store them as advice, there would be little work needed, actually just a little modification to advice storing method as all of the session methods (login, logout) are already implemented and shared across all client types.

7.3.3 User module and advice module

As the names indicate this module contains all of the functionality to manage user information storing and fetching as well as handling login and logout functions. On the other hand similar to user, advice modules handles all of the advices, comments and flags storing and fetching.

7.3.4 Session module

Session module takes care of all the information that needs to be stored during the time period of one user session; it stores all of the user and its profile data so we can indentify user and connect it with its actions.

7.3.5 SQL module

Like described before, our system is database management system independent because we developed this module that abstracts queries and uses corresponding *PHP* functions to make actual queries to database. For time being only *PostgreSQL* subclass is used, but if system one day needs to be ported to *MySQL*, writing different *MySQL* subclass would make that possible.

7.4 Client side modules

Client side modules are much simpler, because like we said before we didn’t want too much functionality placed on client side, and much of the functionality is available to us through Google Maps API, so the only modules here are main module to handle system functions like map displaying, Google Maps wrapper module that abstract functionality available from Google (so that another provider can also be used if needs arise), and modules to check user input.

7.5 Client server communication

All of the client server communication is, like names of nodes indicate, strictly one way. So server delivers all of the data to that client requests. Client can request data in different ways, through forms or through *AJAX* calls. All of the data sent through *AJAX* is coded with *JSON* string so that we can share same data structures both on server and client side which speeds up and automates data exchange; no need of parsing is needed as both Prototype library on client side and *PHP* built-in functions support *JSON* decoding.

One example of client call is shown on *Figure 6* below.

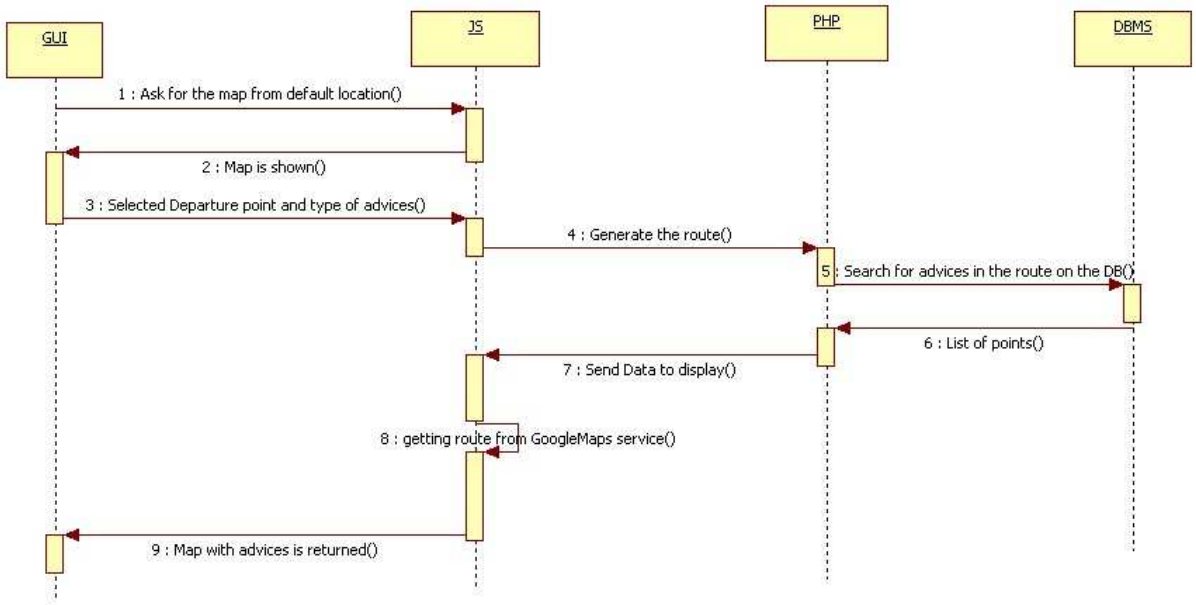

Figure 6, example sequence diagram for displaying advices and route

8 Implementation

8.1 Server implementation

Server code is mostly object-oriented implemented because we wanted to make some rules in somewhat messy *PHP* code in general (based from programmers experience gathered from personal experience and developers on web). This results in minimal speed loss, which in project of this volume is indeed less than noticeable. This object orientation proved to be a great deal of help towards easy implementation because everything about classes and interfaces was known and decided from beginning. It is true, if we decided to use non object-oriented code, the development would be much faster and non-dependent but it would be "a nightmare" to maintain and extend. So with good class documentation we shared knowledge among our programmers. We implemented modules separately and connected them through service layer. Service layer makes use of these modules by invoking different modules methods that they provide. Each method from module followed strict rules on interface that we have defined in the design phase and posted on our *BTW Google group* as developing guidance.

8.2 Client implementation

Client implementation was somewhat trickier and it required effort from our main programmers due to strange world of Javascript and AJAX. For strange we mean complex to deal with, being that it requires to work with

asynchronous calls. The main problem here was browser differences, but again we succeeded by using a good proven *Javascript* library called *Prototype* [13] that handles *AJAX* browser independent. It is the common known open source Prototype library which we simplified by boxing it in our own *AJAX* client interface that handles all of the calls to server. We also boxed all of the functionality provided from *Google Maps API* thus making our system ready for other providers also.

8.3 Server – Client load ratio implementation

Due to complexity of this project and many technologies being used up, we wanted to take a careful approach to take a decision on which modules should be placed on client side, and which on server side to optimize user experience together with system responsiveness. For example it was our initial idea to put route logic calculation on server side to customize routes as much as possible, but giving it some further thought we realized that this logic was a great candidate for "bottleneck" problems together with database layer since it would require of system to take constant queries to maps service provider.

Since the early project status we decided to go with Google Maps API in which we already had that logic implemented as internal part of API. This was a great deal of help from Google for our project since we had strict timetable. The bottom side of this is lack of route optimization for given advices at the time being since Google Maps API for now doesn't provide ways to customize routes, this is why we abstracted the whole map layer to be opened for other map providers, even if we see Google as the most potential one.

Another thing regarding the load balance is the thing we did not want to avoid, and that is AJAX calls. Since this is a web application, we wanted not to stress users with constant page reloading, and even in some cases it was impossible to complete project without AJAX calls like in fetching advices information directly from map.

8.4 Outcome

The final product resulted with over 800 lines of PHP object oriented code, more than 1500 lines of user interface code, and about 1200 lines of Javascript code totaling with 3500+ lines of code. Maybe it is good to mention that this was all code written by us, and the total amount of code grows over 10000 lines together with libraries used.

9 Verification and Validation

To verify that system is bug-free and it is working properly we used "unit tests" as well as global tests. Unit test were performed by the developer who implemented some service methods and they were performed on the server side code without considering client calls. Instead of that calls were simulated with test data to see that the server methods behave properly and bug-free. Global test were performed online with team members that were in charge of testing. They would initiate some random user mistakes and see how the system is responding to them. These mistakes were documented in our System Validation document together with observed system behavior.

The online *W3C Markup Validation Service* was used to check our html code, and *JSONLint*, also an online service was used to insure that JSON data string was correctly formatted. We also used *JSLint* as Javascript code validator.

For each project integration it was required to verify that intended functionality is met properly. That's why we have listed all of the intended functionality and added them to System Verification document, if there were some non wanted results, discussion is raised on group so that the problem can be located with author of specific code section. In *Figure 7* we show only a small part of this document.

Action	Expected result	Actual result	Verdict
Log in	The map is centered on the users' address	The map is centered on the users' address	OK
Add Advice	Data are inserted in the Database and marker is shown	Data are inserted in the Database and marker is shown	OK
Add comment	A new comment is added in the db and shown below the advice in the popup	A new comment is added in the db and shown below the advice in the popup	OK
Remove comment	The comment should be deleted from the db and removed from the popup	The comment is always present	FAILED, FIXED
Log in	The map is centered on the user's address	The map is centered on the user's address	OK
Add flag	The flag is inserted in the db with a comment but not shown to user	The flag is inserted in the db with a comment but not shown to user	OK

Figure 7, Use cases for validation

After the Verification process we validated the system asking the customers if the product fulfills their requirements. We figured out our system is complete and provides all the things necessary to satisfy our stakeholders. We also created a tutorial to help users and now we are working on adding guidance directly in the *BTW Web Site.*

10 Development Process

The main problem we faced starting the project was the short time available since the courses duration is only half semester (although with a double efforts assumed it was much tougher than run the project during one semester with a "normal" effort. While started with a sequential Waterfall-like model in the first planning, (We tried to analyze all the things in detail in the early phase of the project, to decrease the number of future modifications) we continued with an *Agile development process*, to get advantages from some of its features: we had to be adaptable of changing circumstances. To make the project ready to be modified at any moment, we divide it in components, like modules and services. In this way it was possible to modify something in a module without affecting other parts.

Moreover we used weeks as time periods to define deadlines (and we prioritize frequent release of fully working parts instead of working on all the features together.

To develop and provide good quality software we worked iteratively. In each iteration we developed code locally and tested as unit, and then we integrated it on the server, making it available for other developers and testers. Other people can found bugs in the unit or integration problems, so they can notice it to the developer who is supposed to fix them. Once the unit is fully working, a new feature is developed in a new iteration. Finally we executed system testing to check all the units together.

One of the main things to follow was the communication with customers, to know what they exactly need, how they want to get information and so on. To do this, we created a survey to poll people of different age, different gender, from different countries etc.

We could not have face-to-face conversation because of the distance, but to improve communications we decided to have frequent meetings in both Zagreb and Västerås, live meeting once a week among all the members and everyday conversation between the two leaders.

11 Outcomes

11.1 Implemented modules

Entire application consists of several server and client modules; we implemented modules for managing users, advices, service module, and administration module for system administrators. On client side we implemented *Google Maps* module for retrieving maps and routes from Google, AJAX calls module for querying the server and *Map Markers* module for managing all the advices displayed on map.

11.2 The User interface

We took wery big considerations when desgning the user interface. This is completely our design, without using any templates available on internet. We wanted to make an original touch, something rarely seen before and not something that is already available as a template on web and used many times before.

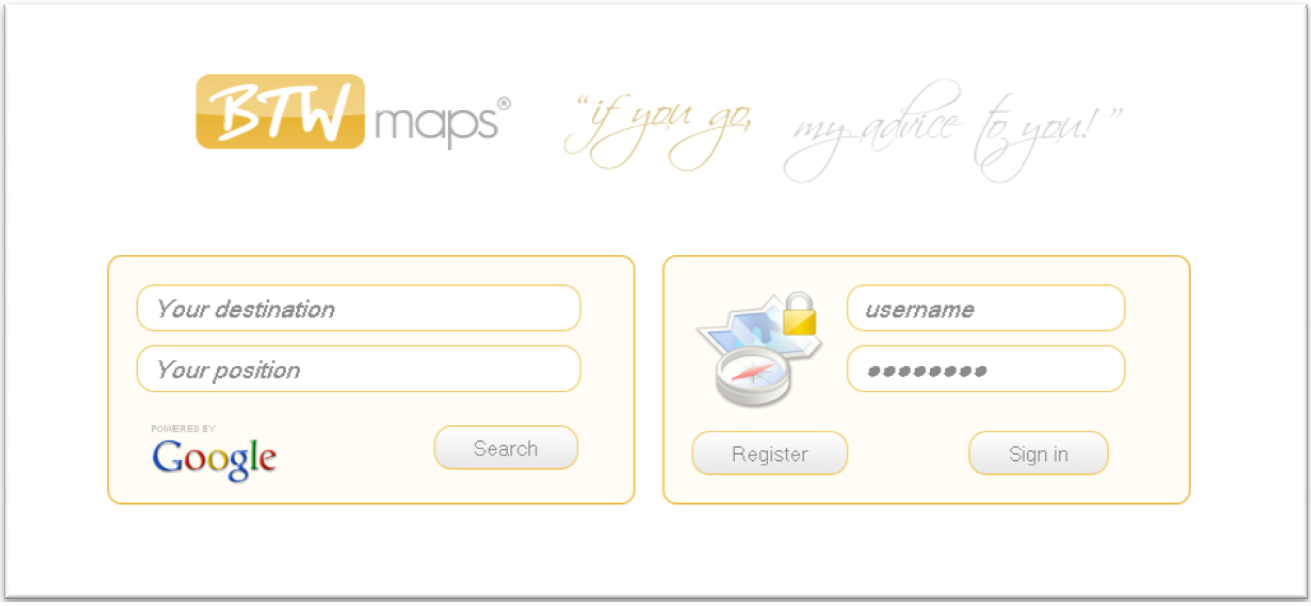

Figure 8, Welcome screen

In *Figure 8,* but also online on our project Web site (http://btw.rasip.fer.hr), you can see how we designed the Home page. It is divided in two parts; on the left user can find routes and advices without registration, on the right the log in permits to access extra features like add advices or choose which categories of advice to show. In *Figure 9* you can see how a typical route looks like and how the advices are showed around it. So when user enters source and destination a route gets displayed together with desired advices. Each of the advice, when clicked on, opens a bubble with details about that particular advice like shown in *Figure 10.* This bubble also contains properties of an advice so users know if advice is e.g. wheelchair accessible or has parking. Users can also comment these advices if they wish and each advice can be commented any registered user. In addition to commenting, users can also flag advice as an inappropriate but they must enter a valid reason for doing so. These flagged advices are then show in administration area so that system administrators can decide whether they want to delete them or not.

Figure 9, My desired route with advices

A user also gets functionality for changing advice or deleting it (if he/she owns the advice of course), and also we can see how easy it is to filter advices simple by clicking and colorizing them on left filter menu. So basically it is very intuitive and powerful, so searching for all "wheelchair accessible" ATMs on your route is just few clicks away! This is possible because we made a distinction between advices category and properties (*Figure 10*). Everything is categorized in two ways, what the advice actually is and what properties it has - the properties such as working time, accessibility, “drive in”, etc… and to expand the system even more all of this categories and properties can be customized for each user, and there is way for administrators to easily add new categories and properties also.

These advice properties can be very useful for disabled people because with adding properties like wheelchair accessible, number of steps and similar, they can filter out all those properties that don’t feel their need and they can easily discover e.g. which underpasses are wheelchair friendly and which are not.

Figure 10, Categories available so far

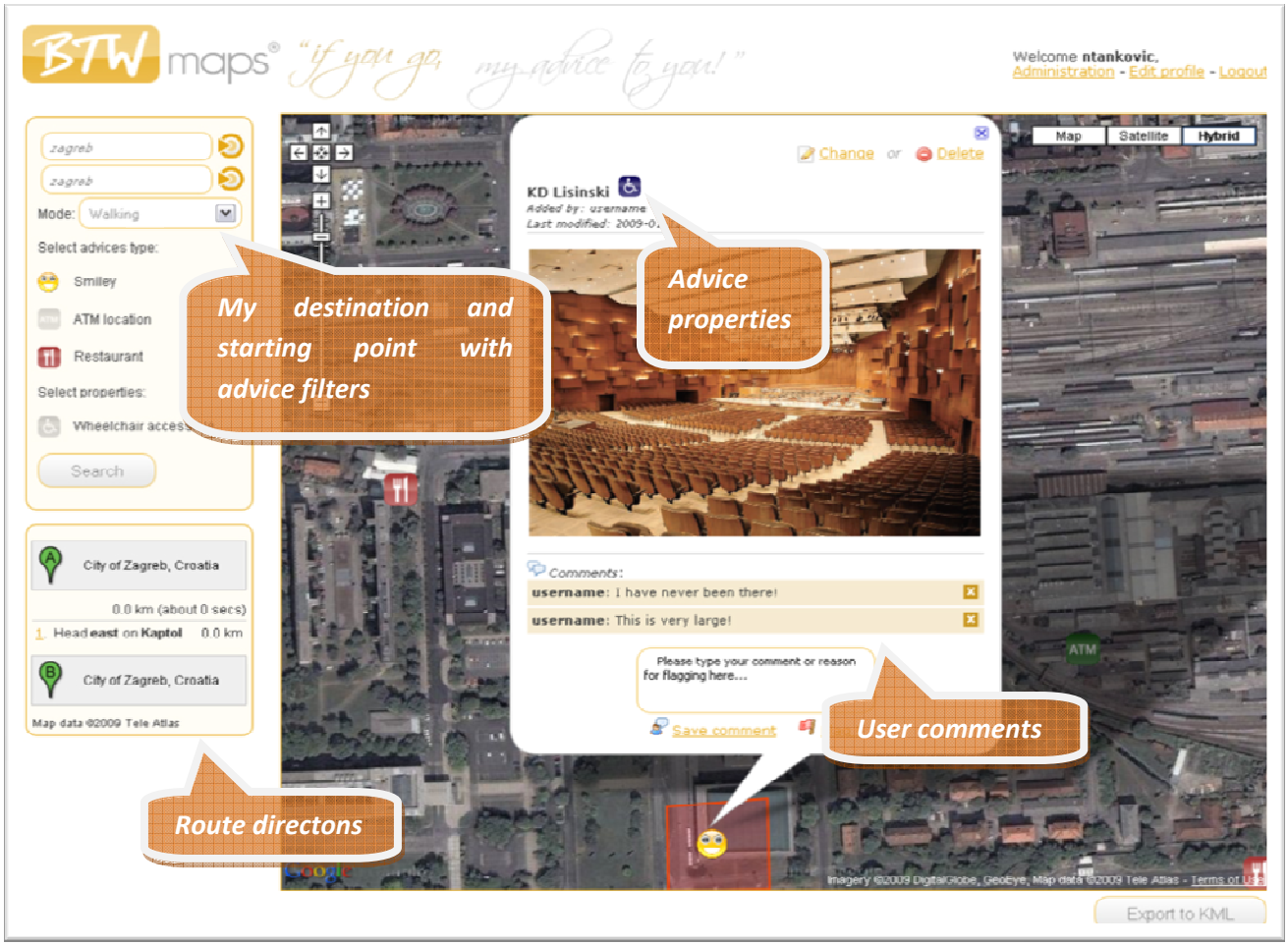

Figure 11, Main screen with visible advice details

11.3 User response to launch

In first few days of our software being available to public, we already have over 30 registered users, and over 70 advices located in Zagreb and Vasteras provided by them (*Figure 12*). They have entered ATM locations, Tram stop locations, Restaurants, Gas stations and pharmacies mostly. For start we made available to them 13 advices types (ATM, restaurant, bus, tram and Metro stops, dangerous area, gas station, shopping malls, pharmacies, parking areas, underpasses and flyovers) and 2 advice properties (Wheelchair accessible and parking). In next days to come, we as starting application administrators, will monitor valuable users and if they agree reward them with administrator status also, so they can enter even more categories and properties that meet their needs. This makes our whole application self maintainable by having administrators from all around the world and requires no programming effort to bring new categories and properties.

We feel a special amount of satisfaction seeing that our application is actually used and that users show interest in it, because that was one of our first goals.

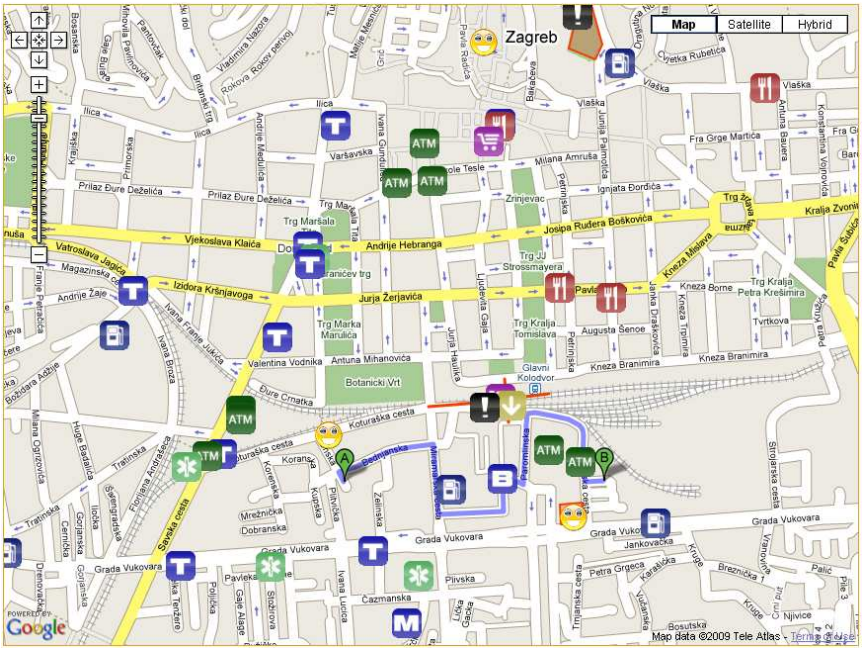

Figure 12, Zagreb full of advices from our new users

11.4 Other outcomes

Other than the final product located at *http://btw.rasip.fer.hr*, we also produced many documents along the way for better project tracking. We have created document regarding project and design description, about requirements and documents containing final project report and acceptance test plan. Each week each of us filled a week report and combining them we created summary week reports. There were also many *Photoshop* [16] files produced containing all the designed artwork including icons and all the pages. The documents are placed in http://www.fer.hr/rasip/dsd/projects/btw/documents.

11.5 Our experiences and lessons learned

This experience was useful for us to learn new technologies and development techniques. We hope this project can be used by people to get information about the places they are planning to go or just when seeking additional information on certain places.

The first lecture we learned is that distributed development is very hard to manage and that it requires a lot of overhead time for conferences in addition to the normal communications and tracking the project. It was quite a shock for us because we developed never before a project with team members few thousand kilometers away from each other, so basically we worked with people we have never seen before. Thus it was very important to set up a friendly environment and team spirit. We are happy that we succeeded in our project, and given as much as we could for it. We are now also all very good friends who are determined to stay in touch and this was an unexceptional experience for all of us on which we thank both our universities and both SCORE supervisors. Taking decision together was one of the most important part of our work, for two different reasons; the first was that some decision were taken in the first period, when we still don't know each other, the second because they were so important to make us save time / work properly for the rest of the project. Moreover this experience was important for the future because we learned new technologies and how to make them work together.

Some of many new things learned as stated by team members:

- PHP programming, it is dynamic and free style other than syntax rigid languages.

- HTML and CSS for designing the web pages
- How to work in layers and how they provide the security and easiness to work within the groups
- Learned some very new and attracting web components and tools like Smarty, Ajax and also some API's provided by the Google

12 Summary

This project was done in a quite short period with an intensive workload (being that it covers half university semester), nevertheless we profuse a lot of effort to provide a quality software. We reach this target working day by day, and through iterations in weekly periods and making different level of tests, both during programming and after deployment. We worked adding modules as "services" iteratively, to have always a working and testable version since the first ones. We also divided the project in layers to make each part as more independent as possible.

Six people were involved in this work, from three different countries and belonging to two different universities, who have had different background and knowledge. The first part of project was the hardest because we had to work hard to line up all the members. After that we proceeded quicker in creating the software and documentation to make the website easy to use (available to all kinds of people, with different requests). During all the developing process we focused on making the product extensible and ready to new technologies, like mobile phones.

The experience was great for all of us, we learned how to work in team in spite of distance and other working problems, and we also found some friends more than co-workers. Software we produced is already being recognized by the users, and with only a few days passed of spreading our word of its existence, we gathered many advices to our database. We have given users a great amount of freedom in structuring their advices which can be polygons, lines or dots and contain multimedia information and links to relevant advice Web page. Administration allows infinite number of advice categories and properties to be added, and with our promotion system, many users can become administrators or moderators thus making information more reliable and useful.

13 References

1. Google Maps – *http://maps.google.com* (10.01.2009)
2. HTML - *http://en.wikipedia.org/wiki/HTML* (10.01.2009)
3. WCAG2.0 web site - *http://www.w3.org/TR/WCAG20-TECHS/* (14.01.2009)
4. Google Maps API - *http://code.google.com/apis/maps/* (14.01.2009)
5. *SOAP* - *http://www.w3.org/TR/soap/* (11.01.2009)
6. *REST* - *http://rest.blueoxen.net/cgi-bin/wiki.pl?FrontPage* (08.01.2009)
7. *JSON* - *http://www.json.org/* (14.01.2009)
8. XML - *http://www.w3.org/XML/* (11.01.2009)
9. PHP – *http://www.php.net* (14.01.2009)
10. Javascript – *http://en.wikipedia.org/wiki/JavaScript* (14.01.2009)
11. MySQL - *www.mysql.com/* (12.01.2009)
12. PostgreSQL - *www.postgresql.org/* (12.01.2009)
13. Prototype library – *http://prototypejs.org* (12.01.2009)
14. AJAX (Wikipedia) - *http://en.wikipedia.org/wiki/Ajax_(programming)* (10.01.2009)
15. iPhone – *http://www.apple.com/iphone/* (10.01.2009)
16. Adobe Photoshop - *http://www.adobe.com/products/photoshop/compare/* (15.01.2009)