



TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

V-CONNECT - VOLUNTEER RECRUITMENT AND MANAGEMENT SYSTEM

LICENSE THESIS

Graduate: **Dariana LUPEA**

Supervisor: **Assist. Eng. Cosmina IVAN**

2018



TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Dariana LUPEA**

V-CONNECT - VOLUNTEER RECRUITMENT AND MANAGEMENT SYSTEM

1. **Project proposal:** The purpose of this project is to design and implement a web information system that provides support for volunteer recruitment and management processes. Instant messaging and notification mechanisms will be implemented, in order to facilitate the communication between volunteers and organizations.
2. **Project contents:** Table of contents, Introduction, Project Objectives, Bibliographic Research, Analysis and Theoretical Foundation, Detailed design and implementation, Testing and Validation, User's manual, Conclusions, Bibliography, Appendices.
3. **Place of documentation:** Technical University of Cluj-Napoca, Computer Science Department
4. **Consultants:**
5. **Date of issue of the proposal:** November 1, 2017
6. **Date of delivery:** July 9, 2018

Graduate: Dariana LUPEA

Supervisor: Assist. Eng. Cosmina IVAN

**TECHNICAL UNIVERSITY**

OF CLUJ-NAPOCA, ROMANIA

**FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT****Declarație pe proprie răspundere privind
autenticitatea lucrării de licență**

Subsemnatul(a) _____

legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării

_____ elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea _____ din cadrul Universității Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data

Nume, Prenume

Semnătura

Table of Contents

Chapter 1. Introduction	3
1.1. General context	4
1.2. Project context.....	5
1.3. Project content.....	5
Chapter 2. Project Objectives.....	7
2.1. Main objective.....	7
2.2. Specific objectives.....	7
2.3. General objectives	8
Chapter 3. Bibliographic Research	9
3.1 Short history of volunteering	9
3.2. Volunteer management models.....	9
3.3. Volunteer management systems.....	10
3.3.1. Challenges in developing a VMS	10
3.3.2. A reference model for VMS	11
3.4. Similar systems	13
3.4.1. Classification of VMS	13
3.4.2. Analysis of similar system.....	14
3.4.3 Comparative analysis of studied systems	16
Chapter 4. Analysis and Theoretical Foundation	18
4.1. Requirements.....	18
4.1.1. Functional requirements	18
4.1.2. Non-functional requirements.....	19
4.2. Use cases	20
4.2.1. Actors	21
4.2.2. Use case model	21
4.3. Conceptual architecture of the system	28
4.4. Technological perspective.....	29
4.4.1. Spring framework	29
4.4.2. Other Spring Projects	31
4.4.3. Apache Maven.....	32
4.4.4. JPA (Java Persistence API)	33
4.4.5. JSON Web Token (JWT)	33
4.4.6. RESTful Web Services.....	34

4.4.7. MySQL	34
4.4.8. HTML, CSS and Sass	35
4.4.9. Bootstrap.....	36
4.4.10. AngularJS	36
4.4.11. Node package manager (npm), Bower and Grunt	38
4.4.12. WebSockets	39
4.4.13. Google Geocoding API	40
4.4.14. Git, Bitbucket and SourceTree	40
Chapter 5. Detailed Design and Implementation.....	42
5.1. Web application architecture.....	42
5.1.1. General description.....	42
5.1.2. Components description	42
5.2. Backend Architecture	46
5.2.1. General description.....	46
5.2.2. Components description	47
Chapter 6. Testing and Validation	58
6.1. Manual testing	58
6.1.1. Test case 1: Organization creates an event.....	58
6.1.2. Test case 2: Volunteer applies to event	59
6.2. Automation testing	59
Chapter 7. User's manual	62
7.1. System requirements	62
7.2. Installation and running.....	62
7.3. Utilization instructions	64
7.3.1. Sitemap	64
7.3.2. Application navigation	65
Chapter 8. Conclusions.....	72
8.1. Achievements	72
8.2. Further development	73
Appendix 1: List of figures.....	77
Appendix 2: List of tables	79
Appendix 3: Glossary	80
Appendix 4: UML Class diagram.....	81

Chapter 1. Introduction

Wherever you turn, you can find someone who needs you. Even if it is a little thing, do something for which there is no pay but the privilege of doing it. Remember, you don't live in the world all of your own. ~ Albert Schweitzer

He has a right to criticize, who has a heart to help. ~ Abraham Lincoln

Volunteering is the ultimate exercise in democracy. You vote in elections once a year, but when you volunteer, you vote every day about the kind of community you want to live in.

~ Author Unknown

Volunteers don't get paid, not because they're worthless, but because they're priceless.

~ Sherry Anderson

Volunteering represents an opportunity of contributing to community. As expressed by the quotes presented above, helping the ones in need denotes an act of kindness and altruism, which is invaluable. Starting with the 21th century, volunteer work became more popular and many organizations were born in order to address problems raised by human needs.

Nowadays, volunteering is of indisputable importance in terms of social progress, therefore an increasing number of organizations and volunteers are collaborating towards achieving meaningful results. As expected, the development of this activity started to generate management issues, since the organizations had to manage lots of people. Moreover, finding volunteers to match specific requirements (skills, experience and competences) was not an easy task.

Together with the Information Technology (IT) evolution in the last few decades, application software development increased, impacting in a positive way different industries such as computer industry, financial services, telecommunications, education, health care, aerospace industry, etc. Organizations began to use different application software in order to improve and efficientize their internal processes.

Volunteer management software such as **eCoordinator**¹, **VolunteerKinetic**², **VolunteerHub**³, **CERVIS**⁴ provide features and functions for solving the most common issues identified in the recruitment, management and measurement processes conducted by organizations.

The proposed project is a web application aiming to provide an interactive and user-friendly solution to the addressed challenges in volunteering, trying to cover two perspectives: volunteer and organization.

¹ <https://samaritan.com/>

² <https://teamkinetic.co.uk/>

³ <https://www.volunteerhub.com/>

⁴ <https://www.cervistech.com/>

1.1. General context

As mentioned in the introduction, volunteer related tasks such as recruitment and management are of high importance, since they compose the mandatory mechanism to accomplish the projects initiated by different organizations.

Many types of volunteering activities may be distinguished worldwide, based on their context⁵:

- **Skills-based volunteering:** Nonprofit organizations choose their volunteers based on skills, competences and talents in order to achieve their missions.
- **Volunteering in developing countries:** This type of volunteering is of high interest among young people, who are travelling to communities from developing world to work on different projects, such as: English teaching, medical work, assisting non-governmental organizations.
- **Virtual volunteering:** Also called online volunteering or e-volunteering, it is done remotely using the Internet and a computer to complete tasks.
- **Micro-volunteering:** Is performed via an internet-connected device.
- **Environmental volunteering:** It refers to the volunteer activities which are oriented towards environmental management or conservation.
- **Volunteering in an emergency:** Emergencies such as natural disasters (tsunamis, floods, droughts, hurricanes, etc.) require a large number of volunteers, who play a pivotal role in the recovery process.
- **Community volunteer work:** Denotes the volunteering which has as purpose the improval of local communities, through nonprofit organizations, churches or local governments.

In general, despite the place where they are based in, organizations encounter similar challenges in their day-to-day activities, therefore we will focus on the analysis of the results of a Romanian study [1], which measures the impact of volunteering in Romania.

In Romania, based on the above-mentioned study, carried out by VOLUM⁶(the Federation of Organizations which Support Volunteering in Romania), over 20 000 volunteers are annually engaged in different projects and during the last 5 years an increasing number of people got involved. In 2014, the first Romanian online platform for matching volunteers and opportunities was developed (<http://www.hartavoluntariatului.ro>) and also the first volunteer management software application from Romania and Europe was released (<http://www.volunteer360.org>).

The statistics from that study show that the greatest difficulties encountered by the organizations in monitoring and evaluating the volunteers' engagement are:

- **68%** - not sufficient human resources to completely and correctly evaluate those processes
- **41.5%** - the pieces of information are collected through monitoring and assessment tools, but the organization members don't have time to actually process and analyze them

⁵ <https://en.wikipedia.org/wiki/Volunteering>

⁶ <http://federatiavolum.ro/>

- **32%** - not enough knowledge and competences for performing the evaluation

Moreover, the first two reasons why the organization does not evaluate the work of its members are the lack of time and people and because the final results of a project are actually analyzed, not the individual contribution of volunteers.

Another factor of influence is represented by the monitoring tools, which are used by the majority of the organizations (**76.2%**):

- **53.1%** electronic format
- **37.5%** paper & electronic format
- **9.4%** paper format

In **59.5%** of the cases, the volunteer coordinator or manager has also other responsibilities within the organization and just in **33.3%** situations there is a nominated person for this role.

1.2. Project context

Following my personal volunteering experience in organizations and at different types of events, I have encountered situations which could have been managed in more efficient ways.

The causes behind the inefficiency may be added to the ones extracted from the statistics presented above:

- lack of time for monitoring and evaluating volunteers' work
- volunteer managers have also other responsibilities within the organization
- insufficient competences for measuring volunteers' engagement
- the use of monitoring strategies (such as handwritten questionnaires or handwritten feedback surveys) which are hard to centralize afterwards

It can be deduced that a good software solution to the stated problems should efficientize and fully support the work of a volunteer manager, starting from volunteer selection phase to results assessment phase, which is performed after each project/event completion.

On the other hand, another aspect of equal importance is the difficulty of finding volunteer opportunities. Even if the organizations present the projects on their websites, the volunteers still have to somehow land on them or to find other ways to discover activities, projects or organizations matching their preferences. Facebook, which does not need any introduction, is also used by the organizations for posting events, but unfortunately, same as for websites, volunteering opportunities from other organizers cannot be found in the same place.

This means that other discovered issues are the reduced visibility and missing centralization of volunteering projects in the online environment.

Therefore, a web application for managing all volunteer related processes and also offering a common space for visualizing projects coming from several organizations may be considered a suitable approach to cope with the identified issues.

1.3. Project content

Chapter 1 - Introduction - presents the general context of the chosen project theme, as well as an analysis of the results provided by a relevant case study. Moreover,

the challenges encountered in volunteering activities and how they are influenced by technological advance are emphasized.

Chapter 2 - Project Objectives - offers an overview of the main objective of the proposed project and indicates the specific goals through which it will be achieved. Some general objectives which need to be considered when designing, implementing and testing the application are also presented.

Chapter 3 - Bibliographic Research - represents a summary of the information consulted during the research phase of the project. First, a short history of volunteering and two volunteer management models are presented. Then, general challenges encountered in the development of projects which are similar to the proposed one are described, together with a framework which serves as a reference model. Next, a classification of the studied similar systems is done and their comparative analysis is highlighted into a table.

Chapter 4 - Analysis and Theoretical Foundation - introduces the functional and non-functional requirements of the proposed system, the use case model, as well as the technologies, frameworks, tools and APIs used for transforming them into a final product. The motivation behind the technological choices with their advantages and disadvantages compared to alternatives is emphasized. An overview of the system is offered by the conceptual architecture diagram.

Chapter 5 - Detailed Design and Implementation - shows step by step the decisions made on the project development process, concerning the algorithms and the general data flow. The front-end and back-end system architectures are presented, together with their main components. The database diagram and the database model are also detailed.

Chapter 6 - Testing and Validation - is a summary of the methods used for testing and validating the developed application. Manual testing techniques, as well as automation testing methods are presented. Data validation mechanisms represent another aspect discussed here.

Chapter 7 - User's manual - presents the software and hardware resources required in order to configure the developed system. Installation and setup steps for running the application are provided. The user guide containing informative images can also be found in this chapter.

Chapter 8 - Conclusions - represents a summary of our achievements, by analyzing the obtained results, related to the objectives of this project. In addition, ideas for further development will be presented, together with possible solutions for implementing them.

Chapter 2. Project Objectives

This chapter examines the **main objective** of the proposed project, as well as other **specific objectives**, which need to be accomplished as parts of the main objective. Some general aspects, having as target the project's quality, will also be emphasized.

2.1. Main objective

The main objective of this project is to design and implement a web information system as an interactive and powerful tool for improving the volunteers selection and management processes, as well as facilitating the communication between organizations and associated volunteers.

2.2. Specific objectives

The main project objective is actually composed of several smaller objectives of equal importance, whose accomplishment leads to obtaining the final product. These specific objectives are the following:

- **Registration of new users:** any user will have the possibility to access the application as a guest in order to use the basic provided features, but registration is required for gaining full access. The registration will be differentiated based on user type (volunteer, organization).
- **Authentication:** registered users will be able to sign in and personalize their profiles, as well as to interact with other authenticated users and access all available functionalities provided by the system.
- **Event posting:** this scenario allows organization users to create volunteering events, to which interested volunteers may apply. Events can be filtered by category, date or searched by title. A detailed event page can also be consulted by all users.
- **Participation to events:** represents a direct way used by volunteers to get involved in the volunteering projects. The organizations accept or reject them, based on the information collected from their personal profile pages.
- **Stories section:** volunteer users will be able to add "stories" to their profiles. A story is a short description of their volunteering experience to a certain event and it will be public (visible to all authenticated users).
- **Event management and evaluation:** volunteer management per event is one of the main purposes of the proposed application. Both management and evaluation actions will be done by the organization user, through tasks creation and their assignation to accepted volunteers, monitoring volunteers' progress and assessment of their work, expressed through feedback.
- **Personal space management:** each authenticated user will have the possibility of managing its personal space, through the profile editing functionality. Moreover, the organizations will present events on their profiles, while volunteers will have their stories displayed.

- **Comments section:** each page presenting the event details will have a comments section, where users can discuss about the event. The admin user is responsible with comments section management.
- **Communication and interaction:** authenticated users will communicate by exchanging messages on the platform (volunteer-volunteer/volunteer-organization/organization-organization users). Emailing is another feature used for notifying users of different important events such as successful registration.
- **Notifications:** volunteers will be notified with regards to their position within the events they applied to (when they were either accepted or rejected) and also about the assigned task.
- **Reporting tools:** the reports play an important role in assessing the results after each event completion. Therefore, the organizations will have the option of generating charts and reports in different formats.
- **User accounts management:** the administrator user is responsible with locking the accounts of the users who add inappropriate data to their profiles or leave comments which are evaluated as improper.

2.3. General objectives

In order to fulfill the requirements of each specific objective, the choices related to project development phases need to meet high quality standards. Therefore, the main aspects that will be considered are the **usability** enhancement through an intuitive user interface, the application's **responsiveness** to user requests, as well as the **security** of user data and of resources which can be accessed by unauthenticated users.

Chapter 3. Bibliographic Research

This chapter provides an overview of this project's context, starting with a short paragraph about how volunteer work became popular, which were the first organizations and continuing with the volunteer manager's role and the responsibilities it involves.

Another important aspect outlined in the current chapter is represented by the challenges met when developing a volunteer management system (VMS). A reference model, which could be used as a framework for building up a VMS, will also be described.

Moreover, a personal classification of the systems which have similar functionalities to the proposed system is indicated, together with a comparative analysis, based on their features and functions. The comparison emphasizes the difference that our project aims to make.

3.1 Short history of volunteering

Volunteering is generally considered an altruistic activity where an individual or group provides services for no financial or social gain "to benefit another person, group or organization" as described in the paper [2]. From a historical perspective, the term was first recorded in 1755, but used since c. 1600, when the noun *volunteer* denoted "one who offers himself for military service".

During the 19th century, younger American people started helping the needy in their communities. In 1881 the American Red Cross was founded and began mobilizing volunteers for disaster relief operations. In the first few decades of the 20th century, several volunteer organizations were founded, including the Rotary International, Kiwanis International, Association of Junior Leagues International, and Lions Clubs International⁷.

After World War II, people started to be interested in volunteering overseas, using World Wide Web⁸ for finding new opportunities.

3.2. Volunteer management models

Volunteer management became an occupation during the late 1960s and adopted practices such as job descriptions and interviews in order to find suitable volunteers for the organization's needs. Nowadays the role of volunteer manager ranges from defining volunteer job roles and recruiting volunteers to organizing, supervising and motivating them, as emphasized in the article [6].

As presented in the article *Models of Volunteer Management: Professional Volunteer Program Management in Social Work* [3], there are some universal volunteer management models. Next, we will shortly present two of those models, which are relevant in this context: the **universalistic** volunteer management model and the **conditional** volunteer management model.

⁷ <https://en.wikipedia.org/wiki/Volunteering>

⁸ https://en.wikipedia.org/wiki/World_Wide_Web

A. Universalistic Volunteer Management

This model highlights the need of nine functions for developing successful volunteer programs: planning and administration, volunteer work design, recruitment, interviewing and screening, orientation and training, supervision and liaison support, ongoing motivation and recognition, impact evaluation, recordkeeping and reporting. Practitioners with universalistic perspective adopt a conception known as “one size fits all” of volunteer management.

B. Conditional Volunteer Management

In contradiction to the model presented above, the current model states that effective volunteer management is influenced by aspects such as the involvement of paid staff in the organization, grassroots membership and organizational size.

There are two types of factors in the conditional volunteer management: volunteer focused and program/organization-focused. Four models of involvement proposed for the first type of volunteer management model may be distinguished: service delivery, support role, member/activist, and co-worker. The second model is centered on the volunteer program type and it assumes that volunteer management is influenced by the organization’s culture and worldview.

The volunteering management models described above are relevant in the context of the proposed project, since they offer an overview of the most important aspects taken into account by volunteer managers in their activities. The factors indicated by each model influence the management process, therefore they will be considered when developing the proposed system, by mapping them to several features and functions.

3.3. Volunteer management systems

According to the paper *A survey on Volunteer Management Systems* [4], information and communication technology (ICT) represents a major enabler of volunteering process. Volunteer management systems (VMS) are useful tools as they facilitate volunteer matching with volunteering opportunities, tasks scheduling and allocation, communication and coordination mechanisms, therefore improving the work of Volunteer Involving Organizations (VIO). There are several VMS, but they may be differentiated by their purpose and the specific functionalities for supporting the volunteering process.

3.3.1. Challenges in developing a VMS

The major challenges of the VMS with regards to the volunteering process, as extracted from the paper [4] and [7], will be presented next.

- **Inhomogeneous Conglomerate of Resources and Tasks**

A major challenge is taking into account diverse skills, competences, interests and personalities of volunteers for appropriate task allocation, which in the end influences their motivation and satisfaction. Also, the work should be splitted into tasks of appropriate granularity, which requires *sophisticated workflow execution mechanisms* at

runtime. Another challenge is introduced by the unexpected tasks, because the *workflow should be adapted at runtime* to introduce *ad-hoc tasks* and *add new users*.

- **Flexible Allocation Allowing Brokerage and Negotiation**

Volunteer work is unpaid, so keeping volunteers engaged represents the key. The VSM should enable a kind of *brokerage* between tasks and volunteers (to find the best match between them) and support different *matching strategies*. This means that tasks should not be assigned just by the VIO, but volunteers may choose tasks from the available ones and also propose tasks to other volunteers. The volunteer makes the final decision of undertaking certain tasks and this might require *negotiations* with the VIO.

- **Adaptation- and Motivation-Oriented Assessment**

The expected outcome of a task is not always clear in advance, so the use of assessment mechanisms is crucial in order to improve the process and to find good motivation strategies. The most important aspects to be assessed are represented by: volunteer engagement, tasks, task allocation and satisfaction/gain of expertise.

- **Continuous Evolution**

Volunteering process follows a “learning by doing” approach, being continuously improved. Task’s creation and allocation are targeted as key points in the progress of an organization, but also the awards, ranking and other motivation strategies are considered.

Our system aims at offering a viable solution to the above-mentioned challenges, by taking into account the task creation and allocation processes, as well as the assessment of volunteers’ work and their motivation.

3.3.2. A reference model for VMS

Based on the challenges highlighted in the previous section, we will analyze a reference model (RM) extracted from the article [4]. This RM serves as a framework, which describes the core artifacts of a VMS. The framework’s structure consists of key packages, defined by **VMS’ structural components**, its **process** and **VMS’ evolution**.

A. VMS Components

This package presents the main components which a VMS should have and it can be decomposed into several sub-packages: *Resources*, *Tasks*, *Profiles* and *Relationships*. The resources may be classified as humans (cf. Human Resource) and non-humans (cf. NonHuman Resource). Further on, *Human Resources* could be *Volunteers* or *NonVolunteers* and *NonHuman Resources* may be distinguished into *PhysicalResources* and *VirtualResources*. The tasks may be splitted by ‘humanity’ criteria into *HumanTasks* and *NonHumanTasks*, by ‘virtuality’ into *PhysicalTasks* and *VirtualTasks*, by ‘commonality’ into *CollaborativeTasks* and *CooperativeTasks*, by ‘predictability’ into *PlannedTasks* and *AdHocTasks* and by ‘locality’ into *RemoteTasks* and *OnSiteTasks*. The *Profiles* package represents the connection between *Tasks* and *Resources*, being composed of *Features*.

The figure 3.1 presents in details the components of Profile and Task packages and how are they connected.

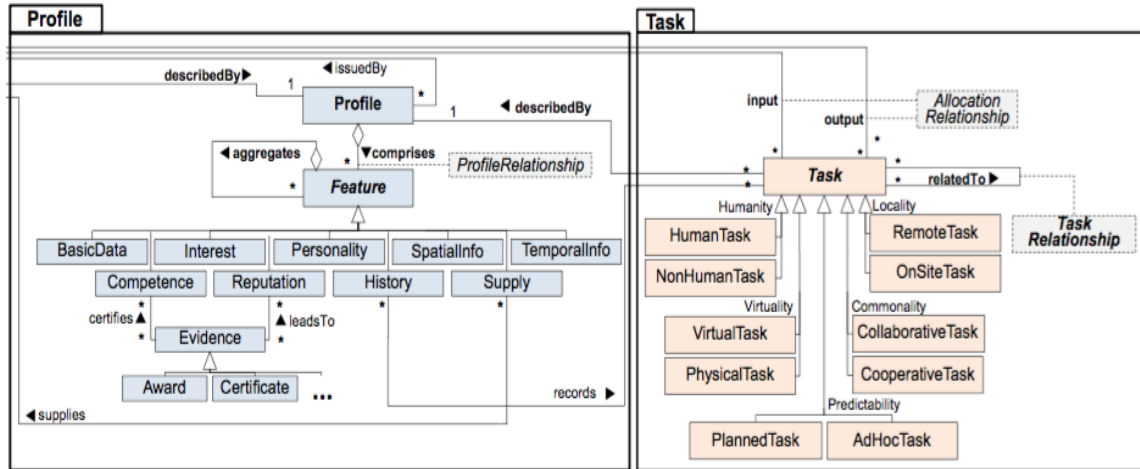


Figure 3.1 Profile and Task packages

B. VMS Process

A VMS system should support the following activities: **Allocation**, **Execution** and **Assessment**. Within allocation and execution phases, tasks should be assigned to volunteers who will execute them.

The assignments may be done manually or automatically by the system and might be *Rejected* by volunteers or *Delegated* to others. The execution progress of a task is marked by its state: In Progress, Suspended, Adapted, Finished or Canceled. Depending on the organization's needs, different Assessment Methods might be required. Moreover, *Self assessment* (when volunteers assess themselves) and *External assessment* (done by others) may be distinguished, based on the assessee's role.

C. Evolution

The *Evolution* package from the figure 3.2 contains the activities required for continuously improving the volunteering work. Therefore, methods of **adaptation** and **motivation** are included in this package. *Appraisal*, *Awarding*, *Ranking*, *IncentiveGranting*, *Challenging* and *Recruiting* are motivational aspects mentioned in the current framework. Concerning adaptation, the competence profile of a volunteer should be adaptable, after the completion of a task.

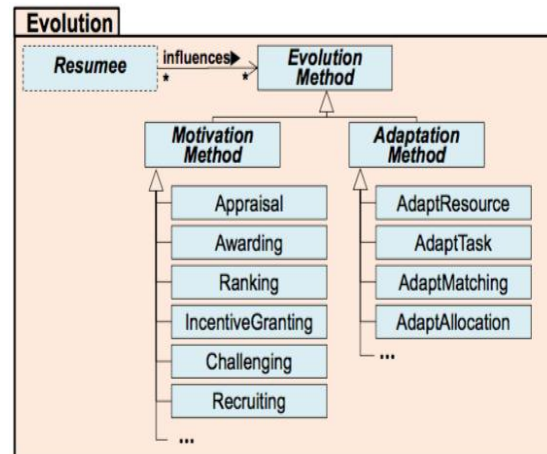


Figure 3.2 Evolution Package

3.4. Similar systems

This subchapter presents a personal classification of the software systems which are used for volunteer recruitment and/or management, based on information acquired during the research phase of the project. Several systems which have similar functionalities will be outlined, as well as a qualitative comparative analysis of those systems, including the proposed one.

3.4.1. Classification of VMS

By studying different systems used in the volunteering field (for recruitment and/or management), we discovered that two classifications may be distinguished. The first one is based on the main objective of each system, whereas the second classification highlights the support level they have for gathering multiple organizations.

It should be mentioned that some software systems described in the section 3.4.2. do not belong to any of the below categories. This can be justified by the fact that they are not oriented towards volunteer management, but they present features and functions which are relevant and useful to the development of this project, due to several aspects which will be discussed next.

A) Based on the **main objective** of VMS, three categories were discovered:

➤ **Presentation of opportunities and volunteer recruitment**

Organizations' websites, where registration forms have to be filled by volunteers are part of this category. Volunteers may also be recruited through social media application software such as Facebook⁹, where events presented by an organization can be posted, offering details to those interested. Another way of presenting opportunities are websites where projects from different organizations are presented. *Centrul de voluntariat*¹⁰ is a web application which can be placed into this category.

➤ **Managing volunteer related data**

Systems having this main purpose just store volunteers' data. Therefore, the volunteer manager is able to fulfill its main responsibilities: assign tasks, send e-mails, manage volunteers' schedule, give feedback, etc. The volunteers do not have direct access to the system. An example of such application is *Volunteer360*¹¹.

➤ **Volunteers recruitment and management**

The majority of the analyzed systems are part of this category. They cumulate the functionalities of the systems from the first and second category, offering a variety of useful features to both volunteer and organization users. A good example from this category is the studied system *TeamKinetic*.

⁹ <https://www.facebook.com>

¹⁰ <http://centruldevoluntariat.ro>

¹¹ <http://www.volunteer360.org>

B) By multiple organization support level criteria, two types of volunteer recruitment and/or management may be distinguished:

➤ **Single organization support**

Such volunteer management systems can be used by a single organization, which manages its members. The organization either uses a standalone application or consolidates its own platform (website).

➤ **Multiple organization support**

Some systems represent a common space for several organizations, which present their projects to all users; therefore the volunteers are able to choose from a wide range of opportunities, depending on their preferences. An example of this type of system is *HartaVoluntariatului*¹².

3.4.2. Analysis of similar system

Systems belonging to categories presented before will be analyzed, based on the features and functions identified in the articles [5] and [10]: **volunteer profile, activity tracking, scheduling, online features, email, reporting and exporting, customization, ease of use**. Moreover, two applications which are not used in the volunteering context (Meetup and Google+), but present useful features for our project, will also be studied.

- **eRecruiter**¹³/ **eCoordinator**¹⁴ are two complementary volunteer management systems developed by **Samaritan Technologies**¹⁵. The first one provides an interface which can be used by volunteers in order to: register, authorize a background check, edit their personal profiles using the provided tools, search for volunteering opportunities and apply to them. An advantage would be the fact that it can be easily integrated into another website. The second web system is oriented towards the organizations' needs by offering the possibility to handle the volunteer profiles, manage their schedules, run different types of reports (PDF or .txt). The email functionality and social media integration (Facebook, MySpace, Twitter) represent a plus. A disadvantage would be the fact that the volunteers are able to find opportunities from only one organization.

- **Volgistics**¹⁶ web system has several strong points such as printed communication, volunteer scheduling functionality and integrated broadcast options. In order to allow the volunteers access their profiles (previously created by an organization representative) and schedule for available opportunities, an additional module called **VicNet**¹⁷ must be used. The volunteers can clock-in and clock-out, check the schedule and fill in time slots only if another module - **VicTouch**¹⁸ - is implemented. Several reporting features are

¹² <https://hartavoluntariatului.ro/>

¹³ <https://support.samaritan.com/hc/en-us/articles/206590816-eRecruiter>

¹⁴ <https://www.capterra.com/p/79211/eCoordinator/>

¹⁵ <https://samaritan.com>

¹⁶ <https://www.volgistics.com>

¹⁷ <https://www.volgistics.com/vicnet.htm>

¹⁸ <https://www.volgistics.com/victouch.htm>

implemented: schedule reports to run automatically (PDF or Excel), awards, batch reports, Excel spreadsheet, etc.

The users of this application can consult the video tutorials from their website, which explain how to make the best of the provided features.

- **TeamKinetic**¹⁹ is an application which supports volunteering, clubs, events and local activities. Volunteers are able to register, join opportunities, log hours, share activities on social media and also receive feedback and awards. The volunteer manager can monitor the applicants, send emails and text messages through the platform and track volunteers on the map. Real time reports can be generated and they can be customized based on the needed data.
- **Volunteer360**²⁰ platform is based on Dynamics CRM technology powered by Microsoft and allows organizations to manage documents and volunteer related data. The admin user can generate useful charts by filtering volunteers by age groups, joining year, projects, geographical area, etc. Other features: edit the volunteers' profiles, schedule their time slots, give feedback and generate reports. This system can be used only by the organization's representative (admin) which means that the volunteers do not have direct access to it.
- **Harta Voluntariatului**²¹ is a web application which presents functionalities similar to the ones proposed by our system. It allows the registration of two types of users: volunteer and organization. The volunteer user has a detailed profile and he can search projects, filter them by using different criteria, apply to projects or save them. The organization user can edit its profile, add new projects, manage the volunteers per project, check the history of the proposed projects. The communication is done through online messaging and notifications.
- **Meetup**²² represents a web application which can be used in order to create *meetups*, attend meetings, see the other participants. It is not related to volunteering idea, but it presents some user interface aspects which could be useful to our system design. The events are presented in a user-friendly manner, describing all the necessary details and offering users the possibility to join the event, leave comments, check related topics, filter events through picking the date from a calendar etc.
- **Google+**²³ is a platform used for social networking which is owned and operated by Google. Some features and functions are: user profile, circles, stream, identity service, community, location, Google local, photography, collections. The news can be filtered by category and the users can share images and videos from other social media environments.

¹⁹ <https://teamkinetic.co.uk>

²⁰ <http://www.volunteer360.org>

²¹ <https://hartavoluntariatului.ro>

²² <https://www.meetup.com>

²³ <https://plus.google.com/u/0/>

3.4.3 Comparative analysis of studied systems

In the table 3.1, a comparative analysis of a part of the above described systems will be presented. The analysis is done based on the features and functions that our system aims to implement (on the first column), with focus on their presence in the studied application software (**YES** - if they are present, **NO** - if they are missing, **DK** – do not know). On the last column, it can be noticed that each position is marked with **YES**, meaning that the proposed system aims at prototyping each of those functionalities.

Table 3.1 Comparative analysis of the similar systems

	eRecruiter/ eCoordinator	Volgistics	Volunteer Kinetic	Volunteer 360	Harta Voluntariatului	Proposed system
Direct interaction of volunteers with the system	YES	Only if VicNet module is integrated	YES	NO	YES	YES
Volunteer management per project/event	YES	YES	YES	YES	YES	YES
Generation of reports/charts	YES	YES	YES	YES	NO	YES
Online feedback and/or awards	DK	DK	YES	NO	NO	YES
Visualization of projects from multiple organizations	NO	NO	NO	NO	YES	YES
Scheduling done by volunteers	YES	Only if VicTouch module is integrated	NO	NO	NO	YES
Broadcast events through online messaging and notifications	NO	YES	YES	NO	YES	YES
Email	YES	YES	YES	NO	YES	YES
Documents management	NO	NO	NO	NO	YES	YES
Social media integration	YES	NO	YES	NO	YES	YES

One of the main objectives of our project, from a functional perspective, is to **allow volunteers to directly interact with the system**. This means that they will be able to register, access their profiles and edit them, as well as to express their desire to join a certain project/event in a simple way. From the above presented systems, only **Volunteer360** does not offer this capability, since only the volunteer manager has access to the application.

The only feature which is exposed by all systems is the **management of volunteers per event/project**. This behavior is expected, since the main purpose of a volunteer management system is to provide efficient mechanisms of organizing the people involved in volunteering activities.

Visualizing projects from multiple organizations on the same platform represents a feature which is not usually encountered in VMS. Since the organizations are using different tools for managing their own members, the focus is on their needs, not on the volunteers' interests and expectations. Therefore, through the proposed system, the volunteers' side will also be considered, thus offering them the possibility to find, in the same place, projects from several organizations. **HartaVoluntariatului** is the only analyzed system which presents this feature.

The opportunity to choose from a list of tasks the one that best suits volunteers' skills, interests and capabilities, offers them enough confidence to accomplish tasks and also the motivation to do their best. Therefore, **scheduling done by volunteers** represents an advantage for both volunteers, who are free to choose their tasks, and organizations, which will have many successfully completed projects, as a consequence. **E-Recruiter** and **Volgistics** allow their volunteer users to perform this kind of operation.

Three of the five analyzed systems (**Volgistics**, **VolunteerKinetic**, **HartaVoluntariatului**) have communication mechanisms such as **online messages and notifications**. This way, the users will efficiently share information and receive updates. For example, the system notifies the assignation of a task. Another way of transmitting news is **sending emails**.

As presented in the subchapter 3.3.1, a major challenge in VMS's development is the *Adaptation- and Motivation-Oriented Assessment*, so keeping individuals motivated and engaged is not easy. **Online feedback** would be the first step to overcome this type of challenge, but not all systems use this approach.

Social media integration (Facebook, Twitter, Gmail) offers a greater visibility of the organization's events, representing a way of promoting them, therefore engaging more people in the volunteering activities. The majority of the presented systems (**eRecruiter/eCoordinator**, **Volgistics**, **HartaVoluntariatului**) are connected to one or several social media environments.

Other auxiliary features, which have the role of improving the recruitment and management processes, are **documents management** and **generation of reports and charts**. The first feature is rarely used, but the second one can be found in the majority of the systems.

As a conclusion, our aim is to build a system which supports both volunteer recruitment and management, by offering support to multiple organizations. In order to accomplish this, the ideas extracted from the analyzed features and functions will be the used as a base for specifying the requirements of the proposed system.

Chapter 4. Analysis and Theoretical Foundation

This chapter introduces the functional and non-functional requirements of the developed system, as well as the use case model. A detailed description of one representative use case for each actor will be presented. Our proposal for the conceptual architecture is highlighted, as well as the technologies used and the motivation behind choosing them.

4.1. Requirements

4.1.1. Functional requirements

Functional requirements (**FR**) present a complete description of how the system will function from the user's perspective. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. The table 4.1 contains the functional requirements of the proposed system, together with the associated users.

Table 4.1 Functional requirements

ID	Functional Requirement Statement	User
FR01	Registration	G
FR02	Authentication	V, O, A
FR03	Event details visualization	G, V, O, A
FR04	Events filtering by date and category	G, V, O, A
FR05	Search for events by name or city	G, V, O, A
FR06	Visualization of user profiles	V, O, A
FR07	Personal profile management	V, O
FR08	Creation of events	O
FR09	Participation to events	V
FR10	Volunteer management per event	O
FR11	Creation of stories	V
FR12	Instant event feedback	V, O

FR13	Communication through messages	V, O
FR14	Notifications	O
FR15	Sharing events on social media	V, O
FR16	Reports and charts generation	O
FR17	Documents management	O
FR18	User accounts management	A
FR19	Administration of comments section	A
FR20	Logout	V, O, A

Note: G=Guest, V=Volunteer, O=Organization, A=Administrator

4.1.2. Non-functional requirements

In contrast to the functional requirements, non-functional requirements (**NFR**) dictate properties and impose constraints on the system. They specify ‘quality characteristics’ or ‘quality attributes’, rather than what the system will do. Next, several non-functional requirements identified for the current system will be presented.

NFR1. Usability. Part of the broader term **user experience** and refers to the **ease** of learning and using a system. The user interface of the developed application should be **intuitive** and focused on continuously offering feedback to users, such that their experience will be a pleasant one, not a burden. **UI responsiveness** will be also considered, since users might access the web application from different devices (laptops, tablets, mobile phones).

NFR2. Security. Refers to the ability to **prevent** and/or **forbid** the access of unauthorized parties. Since our system stores user related data, it should assure **confidentiality**, by restricting the access to some pages based on the user’s **authorization level**. Another important aspect is the **protection** of data exchanged through exposed web services.

NFR3. Performance. The most widely used metric for performance is the **response time** (how long it takes to the system to respond to a request). The value of response time should not exceed **2s** for the most computationally expensive requests and should be under **1s** for all the others.

NFR4. Availability. Represents an indispensable metric because both **response time** and **throughput** are **zero** when the system is **unavailable**. It is expressed as a percentage of the application's run time minus the time the application can’t be accessed by users. A desired rate of availability for our system would be over **90%**, obtainable through covering possible failure scenarios from the beginning.

NFR5. Scalability. The ability to overcome performance limits by adding resources. When the system performance is decreasing, additional hardware could solve the problem. Thus, a scalable system is one that allows us to add hardware and get a commensurate performance improvement. **Vertical scalability**, or scaling up, means adding more power to a single server, such as more memory. **Horizontal scalability**, or scaling out, means adding more servers.

NFR6. Portability. The developed system is a web application, so the portability requirement should be satisfied, because cross browser compatibility is far easily achievable than operating systems compatibility (web application vs. desktop application). This means that the proposed system should run on various browsers (Chrome, Firefox, Safari, Internet Explorer), irrespective of the used operating system.

4.2. Use cases

This subchapter represents an overview of the system's actors and their responsibilities, captured in detailed use case diagrams. For each actor, a representative use case will be described, comprising the name of the use case, primary actor, preconditions and postconditions, the basic flow ("happy" flow) and also alternative flows.

Table 4.2 Actors and responsibilities description

Name	Description	Main Responsibilities
Guest	Non-authenticated user, having only view access rights	Navigation between unrestricted pages of the application and access to several services, for visualizing and filtering of events, registration.
Volunteer	Authenticated user, representing the volunteer entity	Profile management, participation to events, communication through messages, profiles visualization *
Organization	Authenticated user, representing the organization entity	Profile management, creation of events, volunteers' recruitment and management per event, profiles visualization, messaging and documents management *
Administrator	Authenticated user	User accounts management and monitoring of comments area on the event page

* The guest user responsibilities are included

4.2.1. Actors

The proposed system supports four types of actors: guest, volunteer, organization and administrator. A short presentation of each actor can be found in the table 4.2, including a description and their main responsibilities.

4.2.2. Use case model

Use cases represent a type of textual requirements specification, describing the step by step process a user needs to follow in order to achieve a goal, using the software system. Use cases capture all the possible ways the user and the system can interact, together with the aspects that could go wrong along the way, preventing the user from achieving the initial goal.

4.2.2.1. Use case diagrams

The figures 4.1, 4.2, 4.3 and 4.4 represent the use case diagrams for the Guest, Volunteer, Organization and Admin actors, respectively.

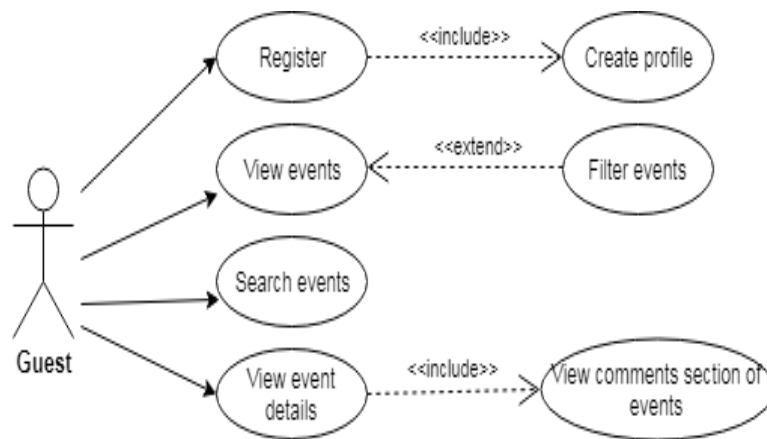


Figure 4.1 Guest Use Case UML Diagram

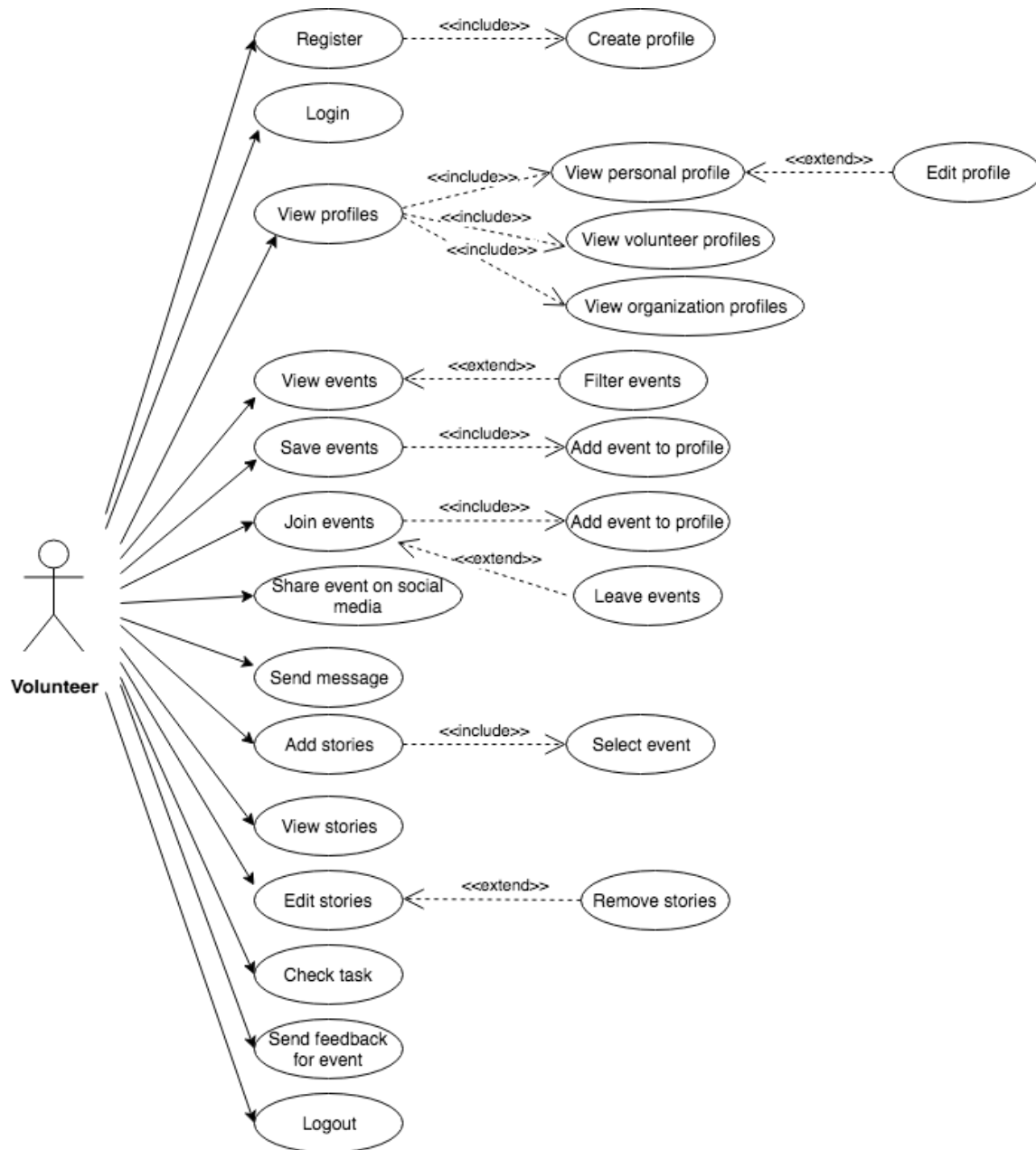


Figure 4.2 Volunteer Use Case UML Diagram

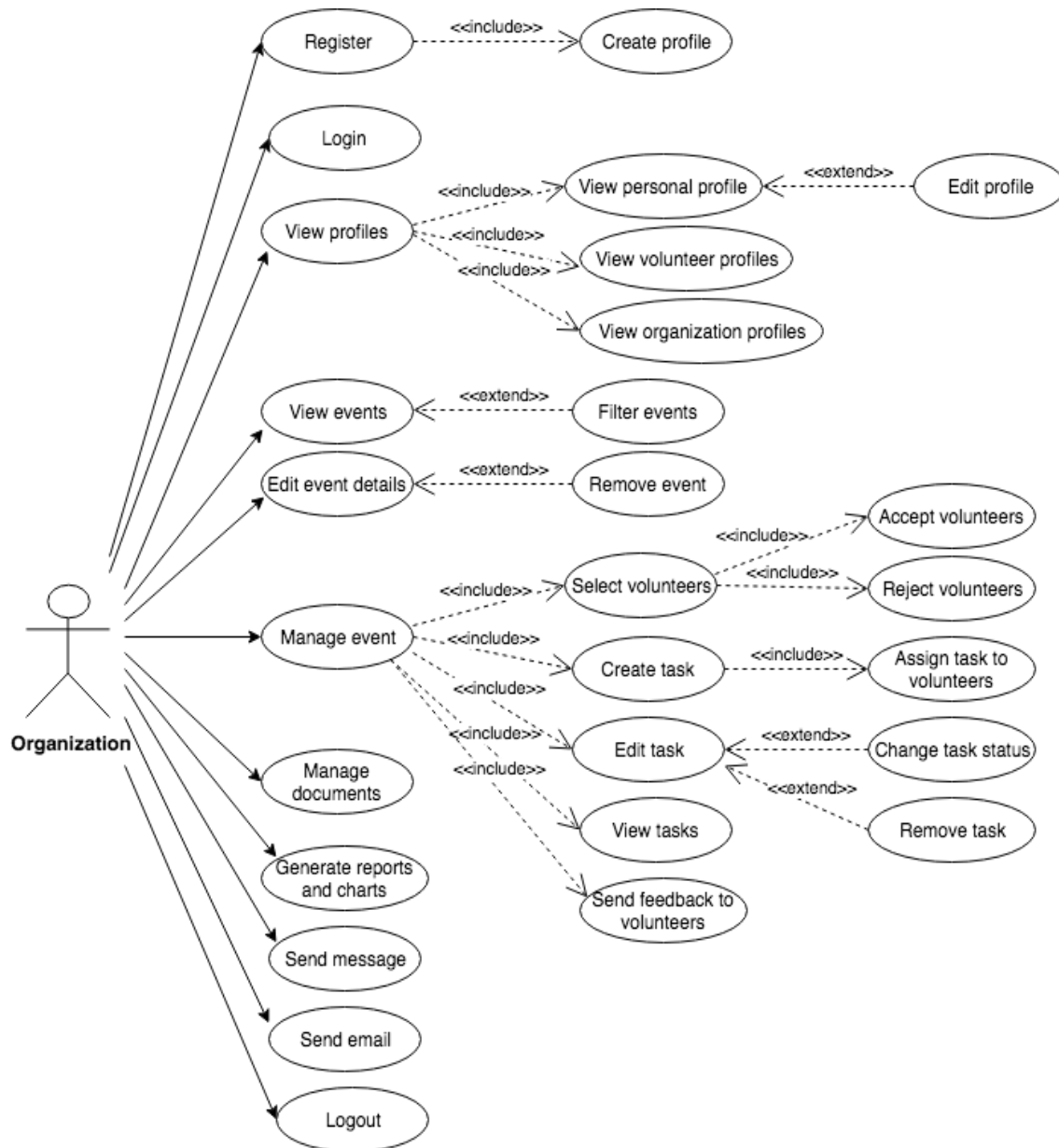


Figure 4.3 Organization Use Case UML Diagram

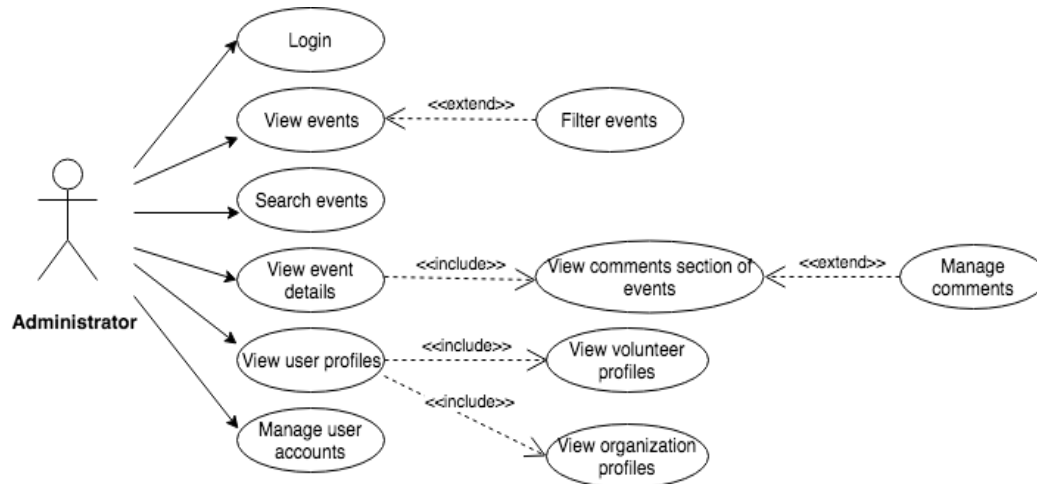


Figure 4.4 Administrator Use Case UML Diagram

4.2.2.2. Use case description

Use case I

Name: Register

Primary actor: Guest

Description: The users which do not have an account are able to perform basic operations such as visualize events, search and filter events and check the details of an event. In order to have access to more complex functions such as profile management, they have to sign up (register).

Preconditions:

1. An account with the same email and/or username is not registered yet

Postconditions:

1. The account is created
2. The information provided by the actor at registration is saved
3. A confirmation of successful registration is sent to the email address provided by the actor

Basic Flow:

1. The actor chooses the register option
2. The actor selects the corresponding type of user: volunteer or organization
3. The system displays a modal presenting the required data: email, username, password, confirmation of password and other personal information, depending on the selected user type
4. The actor fills in the required data
5. The actor finished the registration
6. A success message is displayed

Alternative Flows:

5. The actor does not provide all required data or incorrect input is given
 - a. an error message is displayed and the actor is not able to finish the register operation
 - b. the actor goes to step 4. or abandons the operation

Use case II

Name: Add story about event

Primary actor: Volunteer

Description: After participating at an event, the volunteer user has the possibility to add a story, which appears on his/her profile. A story represents a short description of the volunteering experience and it is public (visible to all authenticated users).

Preconditions:

1. The actor is authenticated
2. The actor is located on the right page

Postconditions:

1. The story is visible on the actor's personal profile
2. Other authenticated users can visualize the story details

Basic Flow:

1. The user selects the add new story operation
2. The system displays the story attributes: title, description
3. The actor fills in the required data presented above
4. The actor selects the event to which the story is related
5. The story is successfully created by the actor
6. The story is displayed on the actor's profile

Alternative Flow

4. There is no event to be selected
 - a. The actor goes to step 5 or abandones the operation
5. The actor's input is invalid (empty fields or no event is selected), therefore the story cannot be created
 - a. The system will signal the error through a corresponding message
 - b. The actor will be asked to enter the story information
 - c. The actor returns to step 3. or abandones the operation

Use case III

Name: Create task for event

Primary actor: Organization

Description: For each event, the organization user creates one or several tasks to which volunteers are assigned. The task is created with “To do” status and the selected volunteers are notified.

Preconditions:

1. The actor is authenticated
2. The actor is located on the right page
3. The event for which the task will be created exists

Postconditions:

1. The task is created
2. The volunteers to which the task was assigned are notified by the system
3. The task can be viewed by involved volunteers and the organization

Basic Flow:

1. The actor selects the event for which the task will be created
2. The actor goes to the task creation section
3. The system displays the task attributes: title, description, location, date, start hour, end hour, volunteers
4. The actor fills in the required data about the task
5. The actor selects one or more volunteers for this task
6. The task is created by the actor

Alternative Flows:

5. There are no volunteers available or less volunteers than needed
 - a. The actor does not select any volunteer or selects just the available volunteers and goes to step 6.
 - b. The actor abandons the task creation operation
6. The input provided by the actor at step 3. is invalid (empty fields) and the task cannot be created
 - a. The system will signal the error through a corresponding message
 - b. The actor goes to step 4 or abandons the current operation
 - c.

Use case IV

Name: Manage comments section

Primary actor: Admin

Description: The administrator user is responsible with the management of the comments section, which is present on the page which displays the details of an event.

Preconditions:

1. The actor is authenticated

2. The actor is located on the page presenting event details, including the comments section
3. At least one comment is displayed

Postconditions:

1. The system updates the actor's changes to comments

Basic Flow:

1. The actor selects the comment from comments section
2. The actor removes the comment
3. The changes are saved by the actor
4. The system displays the updated comments section

Alternative Flows:

At any step, the system can encounter an unexpected error

- a. The actor will be noticed through an explicit error message

The basic flow of events for the third use case - Create task for event – is presented in the figure 4.5.

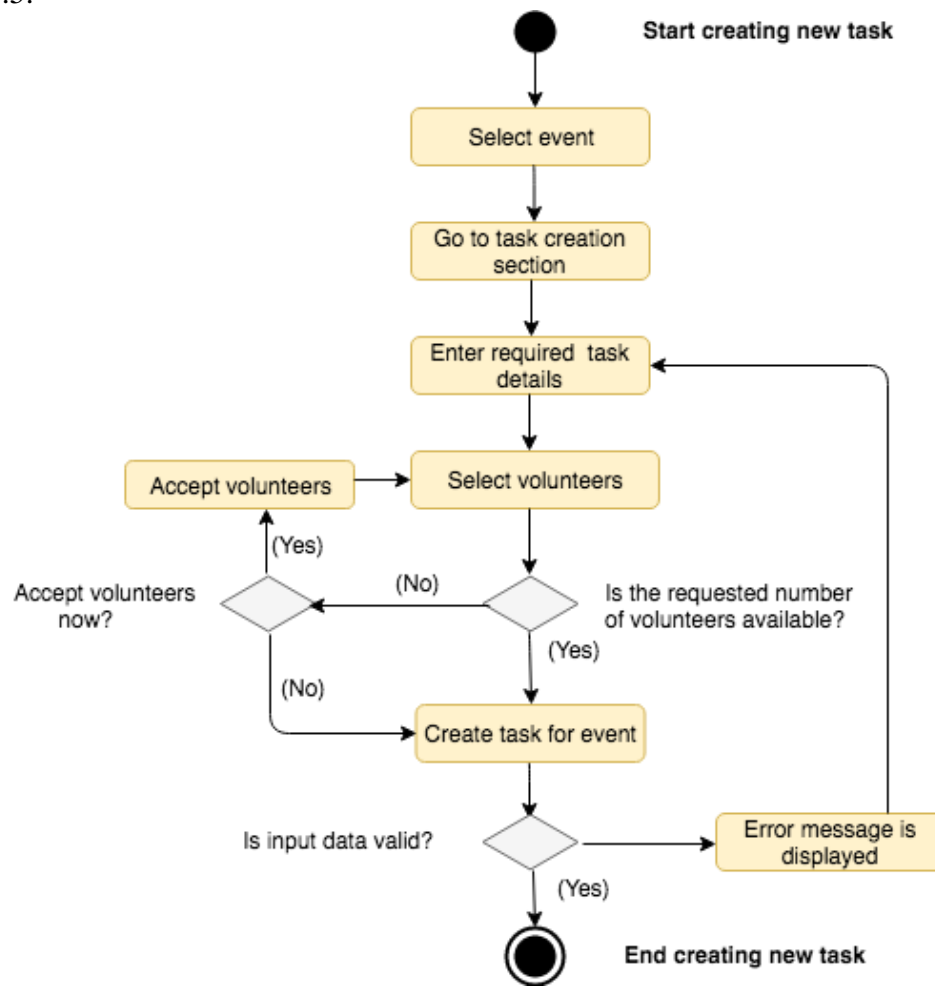


Figure 4.5 Create task use case flow diagram

4.3. Conceptual architecture of the system

In the figure 4.5 an overview of the system architecture is presented. It can be noticed that the client-server architecture is used, therefore the application has a distributed structure, composed of:

- Front-end server (Client)
- Back-end server (Server)
- Database

The first component represents the front-end part of the system and it contains static files (html markup files, css styling files, javascript client-side scripts), used for creating the user interface. The user will directly interact with this part of the application, through a web or mobile browser. This client and the server communicate through HTTP requests and HTTP responses, in order to exchange data.

The second component, which is responsible with processing client requests, is the back-end server. It uses the business logic in order to access and modify the data stored in the database, responding to user requests.

As part of the back-end application, the database is located at the lowest layer, being accessed by the server which establishes a JDBC connection.

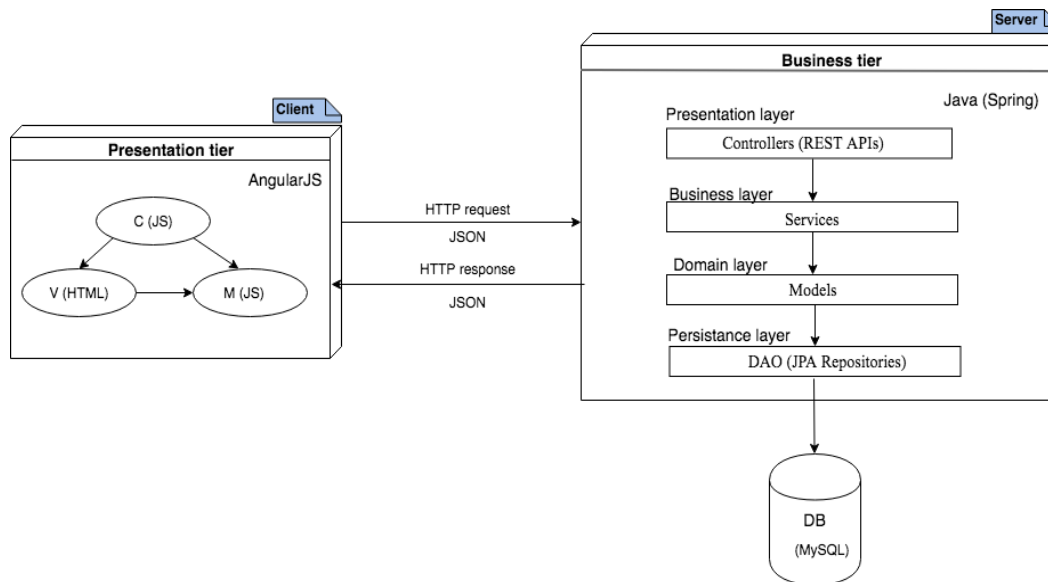


Figure 4.6 System architecture

4.4. Technological perspective

This subchapter presents the technologies, tools and frameworks used for implementing the previously described project components. The value added to our application by each technological choice will be motivated. The main technologies from

our project are also emphasized in the figure 4.5. First, we will discuss the choices made on the back-end side of the project (sections 4.2.1 - 4.2.6) and then the front-end component's technologies will be described.

4.4.1. Spring framework

Spring²⁴ is the most popular application development framework for enterprise Java projects. Millions of developers are using it in order to create high performing, easily testable, and reusable code. Spring framework is an open source Java platform.

Some benefits of using this framework are²⁵:

- Development of enterprise class applications using POJOs, which offers the option of using a single robust servlet container (e.g. Tomcat), instead of an EJB container product such as an application server
- Project organization in modular packages, from which the developer is able to choose only the necessary ones
- Reuse of other technologies such as ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, all in one framework
- Well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks
- Translation of technology-specific exceptions (thrown by JDBC, Hibernate, etc.) into consistent, unchecked exceptions, using a convenient API
- Consistent transaction management that can scale down to a local transaction (using a single database) and scale up to global transactions

As presented above, there are several advantages of using the Spring Framework for the our application's development, since it allows us to focus on the business logic part of the system, rather than on the configuration and integration of other technologies and frameworks.

- **Spring modules**

The Spring Framework includes several modules, which provide a range of services, as presented in the figure 4.6. Some of these modules are: Spring Core Container, Aspect-oriented programming, Inversion of control, Authentication and Authorization, Model-View-Controller, Testing, Data access, Transaction management.

²⁴ <https://projects.spring.io/spring-framework>

²⁵ https://www.tutorialspoint.com/spring/spring_overview.htm

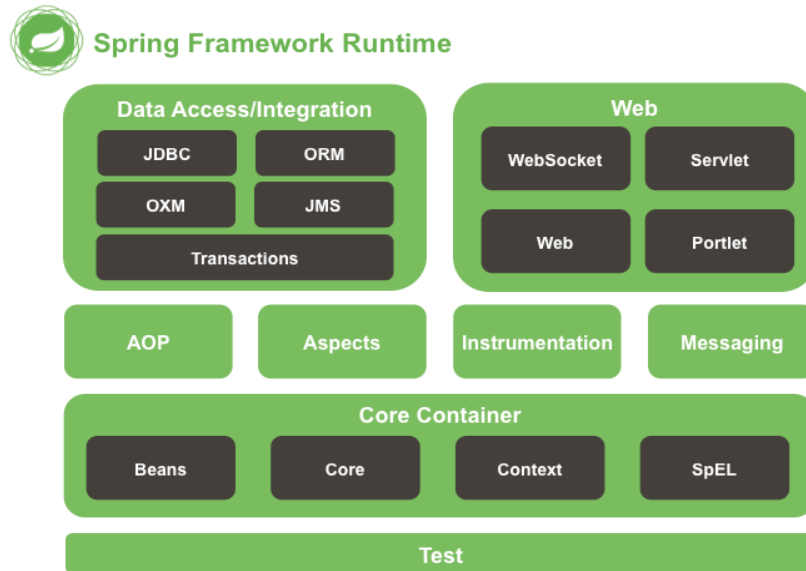


Figure 4.7 Spring Framework overview

The modules which will be used for developing the proposed web application are:

- **Spring Core Container:** It is the base module of Spring and provides spring containers (BeanFactory and ApplicationContext). We used it in our project because of its Dependency Injection²⁶ features, through which the RESTful controllers and services layer on the back-end server side can be connected.
- **Data access:** Spring's data access framework addresses common difficulties faced by developers when working with databases in applications. It provides support for several popular data access frameworks in Java, such as JDBC, Hibernate, Java Data Objects (JDO) and even Java Persistence API (JPA), which will be used in order to efficiently manage the database access.
- **Web:** The web layer consists of the Web, Web-Servlet, WebSocket and Web-Portlet modules. The two way communication between client and server in our web application is implemented using the WebSocket module, on the back-end server side. This technology is important since it provides a communication mechanism between client and server components.
- **Test:** This module supports the testing of Spring components with JUnit or TestNG. The behavior of the proposed system will be tested using JUnit.

²⁶ <http://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

4.4.2. Other Spring Projects

Besides Spring Framework, there are other Spring projects (Spring Boot, Spring Data, Spring Cloud, Spring Security, Spring Web Services and others), which can be integrated into the project, depending on its infrastructural requirements. Next, we will present only the projects which will be included in the web application:

- **Spring Boot**

Spring Boot makes it easy to create stand-alone, production-grade Spring based applications that you can “just run”. Most Spring Boot applications need very little Spring configuration.²⁷

It is a useful technology for us since it simplifies the configuration and initialization steps of the project. One of its main features is the embedding of Tomcat server, which excludes the need of deploying WAR files.

- **Spring Web MVC**

Web MVC Framework from Spring is designed around a DispatcherServlet that dispatches requests to handlers. The default handler is based on the `@Controller` and `@RequestMapping` annotations, offering a wide range of flexible handling methods.

In our project, this module will be used as an “entry point” to the implementation of HTTP methods on the back-end server side. As presented in the figure 4.8, data can be directly returned to clients using `@ResponseBody` annotation.

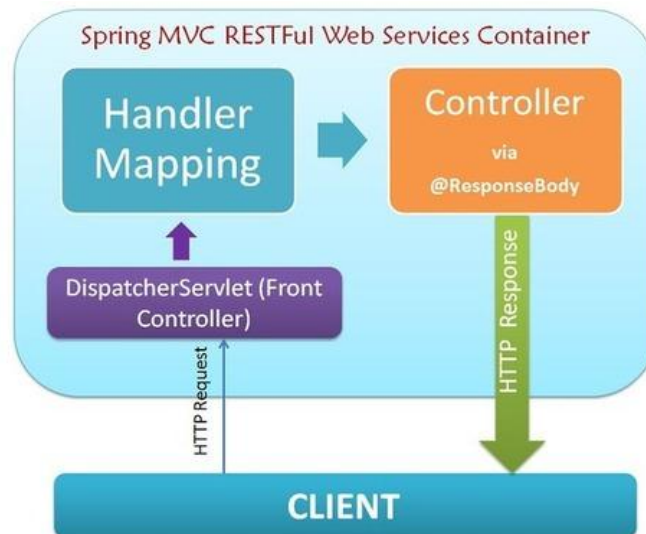


Figure 4.8 RESTful communication in Spring MVC

²⁷ <https://projects.spring.io/spring-boot/>

4.4.3. Apache Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.²⁸

POM is represented through a *pom.xml* file, where the project's name, its owner and its dependencies are specified. When a Maven project is created, Maven builds a default project structure, as shown in the figure 4.9. Individual phases of the build process are configurable as plugins. The build lifecycle is defined by several phases and the standard (default) lifecycle is presented in the figure 4.10.

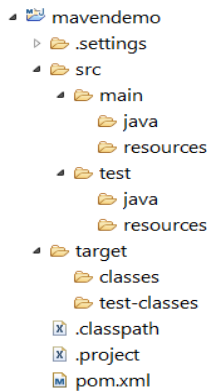


Figure 4.9 Maven project structure

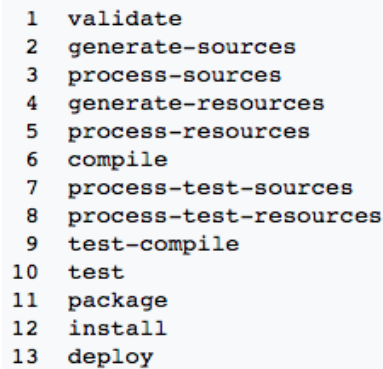


Figure 4.10 Maven's default lifecycle phases

When executing the command **mvn test** for example, Maven runs all goals associated with each of the phases up to and including the “test” phase. Another useful command to new users of a maven project is **mvn install**, which offers the possibility to accurately build, test and install every project.

Several IDEs (including IntelliJ IDEA which we will use) provide integration of Maven, which means that Maven is able to compile projects from within the IDE. Moreover, the add-ons through which Maven is integrated into the IDEs, provide the ability to edit the POM or use the POM to determine a project's complete set of dependencies, directly within the IDE.

Maven is a suitable tool for structuring and managing the back-end part of our project since it automatically adds dependencies in the project, without the need of manually specifying them in the classpath. Another advantage is the easiness it provides in the build management part (life cycle) of the project.

²⁸ <https://maven.apache.org/>

4.4.4. JPA (Java Persistence API)

A persistence entity is a lightweight Java class whose state is typically persisted to a table in a relational database. Instances of such an entity correspond to individual rows in the table.

The Java Persistence API (JPA) is a Java based application programming interface that describes the management of relational data in applications, using Java Platform. In the developed project, JPA will be implemented through several annotations: `@Entity` on Java class (for representing the persisted entity), `@Id` on just one field of the entity (marking the field as primary key) and other annotations on fields such as `@OneToMany`, `@ManyToOne`, `@ManyToMany` for defining the relationships between entities.

Since JPA automatically enables the mapping of entities to database tables, providing “ready to use” implementation of CRUD operations (Create, Read, Update, Delete), it is a choice that simplifies the application’s access to the database.

4.4.5. JSON Web Token (JWT)

JWT is an open standard that defines a compact and self-contained way for securely transmitting information between parties using the JSON format. This information can be verified and trusted because it is digitally signed by using a secret (with the HMAC algorithm) or a public/private key pair using RSA.²⁹

JSON Web Token is used for securing the application’s authentication mechanism and protecting the information exchanged through HTTP requests and responses:

- **Authentication:** After user login, each subsequent request will include the JWT, allowing the user to access only the routes, services and resources that are permitted with that token.
- **Information exchange:** the information is securely transmitted between parties (client-server) using JWT, since the tokens are signed with public/private key

pairs and the signature can be calculated from the header and the payload, in order to verify the content’s consistency.

An efficient authentication mechanism using JWT³⁰ is described in the figure 4.11.

²⁹ <https://jwt.io/>

³⁰ <https://www.toptal.com/java/rest-security-with-jwt-spring-security-and-java>

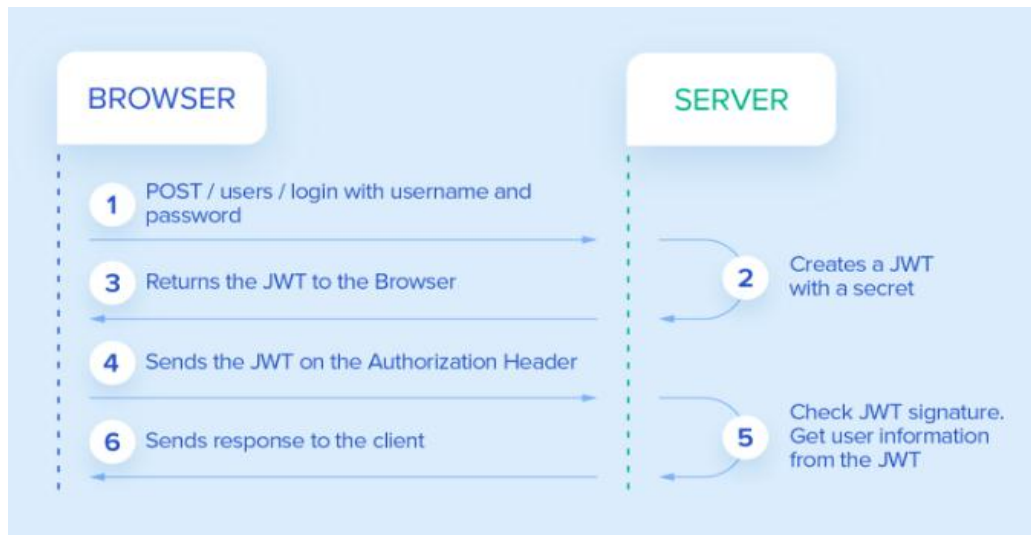


Figure 4.11 JWT security flow

4.4.6. RESTful Web Services

REST (REpresentational state transfer) has become one of the most important technologies for Web applications. REST is an architectural style, primarily used for building Web services that are lightweight, maintainable, and scalable. The main purpose of a web service is to provide access to resources and to allow the client-server communication.

RESTful API is stateless, meaning that each request from client to server must contain all the necessary information to be understood by the server, since storing session state on the server is not allowed. This architectural style is not dependent on any protocol, but almost every RESTful service uses HTTP (Hypertext Transfer Protocol)³¹ as its underlying protocol.

In our project, this type of services will be used for the communication between the frontend component of the application (client) and the backend component (server), using a format recognized by both parties - JSON.

4.4.7. MySQL

MySQL is the most popular open source relational database management system (RDBMS), based on Structured Query Language³² (SQL). All the data manipulated by

the application will be stored in a MySQL database. Some of the reasons behind choosing this type of database are: the fact that it is designed and optimized for web applications and its performance, scalability, reliability, and ease of use are continuously being improved.

³¹ https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

³² <https://en.wikipedia.org/wiki/SQL>

Cross-platform availability of MySQL represents also a plus. Together with this DBMS, another tool used is MySQL Workbench³³, which represents a visual tool for designing, modeling, generating and managing databases.

4.4.8. HTML, CSS and Sass

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.³⁴

HTML pages are composed of HTML elements, which make possible the embedding of images and other objects into the rendered page. The pages are rendered after the web browser receives the HTML documents from a local storage or from the web server. The behavior and content of web pages can be controlled by a scripting language such as JavaScript.

Choosing HTML for building the presentation part of the application was a straightforward decision, because every browser supports this language, it is easy to learn and one can rapidly find solutions to different possible issues.

CSS (Cascading Style Sheets) is a style sheet language used for styling and presenting the documents written in a markup language like HTML. Whereas the HTML is the **meaning** or **content**, the style sheet is the **presentation** of that document. CSS provides the fonts, colors and layout of the text, images, links, tables and many other elements from the HTML pages of the application.

The separation of presentation and content can improve data accessibility and provide more flexibility and control in the specification of presentation characteristics. By specifying the relevant CSS in a separate .css file, multiple web pages are enabled to share formatting, which will reduce the complexity and repetition in the structural content.

Some major benefits offered by this language³⁵ are:

- **Separation of style and structure:** without all the extra HTML for styling the documents' structure, separating the styling into another file makes the code more readable and easier to update, without breaking the document.
- **Faster web page download time:** the use of CSS leads to less code behind the web pages, which means faster download times. CSS code is cached in the browser after the initial request, if it is separated from HTML code.
- **Greater control of presentation:** CSS has more formatting options over HTML such as options to control the spacing and leading of text. One of the most common use of CSS is to present the same page differently to different media.

³³ <https://www.mysql.com/products/workbench/>

³⁴ <https://en.wikipedia.org/wiki/HTML>

³⁵ <https://vanseodesign.com/css/benefits-of-cascading-style-sheets/>

A CSS extension language named **Sass (Syntactically Awesome StyleSheets)** was used, because it adds power and elegance to the basic language, therefore enhancing the application's UI. Other important features of Sass are its full compatibility with CSS, the integration of many functions for manipulating colors and other values and generation of well-formatted, customizable output.

4.4.9. Bootstrap

Initially built by a designer and developer from Twitter, Bootstrap has turned out to be one of the trendiest front-end frameworks in the whole world.

Bootstrap is created for developing responsive projects with HTML, CSS, and JS. Bootstrap 3 supports the latest versions of Google Chrome, Firefox, Internet Explorer, Opera, and Safari (except on Windows) browsers, which makes it even more reliable.

Some of its advantages are the following:

- It is **easy to use**, since integrating it into a project is a simple and fast procedure
- The **responsiveness** of a website has become compulsory, so Bootstrap comes with ready-made classes which allow the components arrangement in the grid system, based on columns
- One of the main benefits is the **speed of development**. Instead of coding from scrape, you can use the existing coding-blocks to assist in setting up. This code can be combined with CSS-Less functionality.
- The **packaged JavaScript components** include functionalities for operating in simple manner tooltips, modal windows, alerts, etc.
- Bootstrap can be **simply integrated** along with other distinct platforms and frameworks, on existing sites and new ones too. One can also use particular elements of Bootstrap along with his/her current CSS.

4.4.10. AngularJS

AngularJS is an open source front-end framework which is based on JavaScript language, being developed and maintained by Google. Its main goals are code simplification and structuring in web applications.

AngularJS is built on the belief that declarative programming should be used to create user interfaces and connect software components, while imperative programming is better suited to defining an application's business logic³⁶.

The main goals of AngularJS are:

- to decouple DOM (Document Object Model) from application logic (the code structure is an important aspect here)
- to decouple the client side of the application from the server side; this allows the development work to be done in parallel, providing reusable components
- offer a structured process of building an application, from designing the UI, through writing the business logic, to testing

³⁶ <https://en.wikipedia.org/wiki/AngularJS>

The main AngularJS concepts are:

- **Directives to extend HTML attributes**

At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's html compiler to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children.³⁷ Such directives are *ngBind*, *ngModel*, *ngClass*.

- **Expressions to bind data to HTML**

Expressions are code snippets used to bind application data to html. They are written inside double braces like `{{ expression }}`. Their behavior is similar to *ngBind* directives and they can be also associated with filters. For example, the line of code from figure 4.12 will display the word “Hello” followed by student’s first name and last name.

```
<p>Hello {{student.firstname + " " + student.lastname}}!</p>
```

Figure 4.12 Example of expression in AngularJS

- **Scope to control variables**

The scope is the “glue” between the HTML (view) and the JavaScript (controller), being available to both. It is an object that refers to the application model and acts like a context for evaluating Angular expressions.

- **Two-way data binding**

Data binding represents the synchronization between the model and the view. It is two-way because when the *model* changes, the *view* reflects the change, and vice-versa. This happens immediately and automatically, making sure that the model and the view is updated at all times. The figure 4.13, which is present in the online documentation of AngularJS³⁸, reflects this relationship.

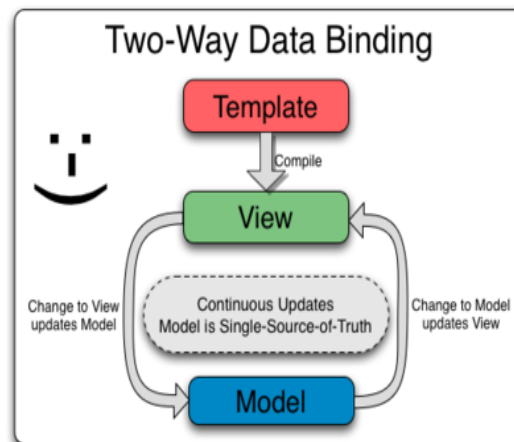


Figure 4.13 Data binding in AngularJS templates

³⁷ <https://docs.angularjs.org/guide/directive>

³⁸ <https://angularjs.org/>

- **Event-handling**

AngularJS has its own HTML directives. They allow running AngularJS functions at certain user events. Some of these directives are: `ng-change`, `ng-click`, `ng-copy`, `ng-keypress`, `ng-mouseenter`, `ng-mouseleave`, `ng-paste`.

- **Controllers**

AngularJS controllers are defined through *ng-controller* directive and have the role of controlling the data of AngularJS applications. They are used for setting the initial state of the `$scope` object and adding behavior to it.

- **Services**

In AngularJS, a service is a function, which is used for organizing and sharing code across the application. AngularJS services are substitutable objects that are wired together using *dependency injection (DI)*. As main characteristics, they are: lazy instantiated (only when an application component depends on them) and singletons (a unique instance is

generated by the service factory function). Services can have their own dependencies, as well as controller do.

There are several advantages of using this framework in our project: it is an open-source platform and it offers flexibility to the developer, making the application easier to customize. Through dependency injection, integrating modules into the project is simplified and two-way data binding facilitates the use of HTML templates.

4.4.11. Node package manager (npm), Bower and Grunt

Npm³⁹ is the package manager for JavaScript, used for installing, sharing and distributing code. Npm is entirely written in JavaScript and its purpose is to manage the dependencies within projects. When used as a dependency manager for a local project, it installs all required dependencies through the *package.json* file.

Bower⁴⁰ is used for managing components that contain HTML, CSS, JavaScript, fonts or even image files. It installs the right versions of the packages one needs and also their dependencies. A manifest file called *bower.json* keeps track of installed packages. It is also optimized for front-end, making sure that all packages are up to date.

Npm and Bower are both **dependency management tools**, but the main difference between them is that npm is used for installing Node.js modules, while Bower manages only the front-end components enumerated before.

Grunt⁴¹ is a JavaScript task runner, which is used for performing frequent tasks in a web application, such as **minification** (the process of removing unnecessary characters from source code without affecting its functionality), **compilation** and **unit testing**. All task runners provide consistency, effectiveness, efficiency and repeatability, but Grunt

³⁹ <https://www.npmjs.com/>

⁴⁰ <https://bower.io/>

⁴¹ <https://gruntjs.com/>

offers access to many predefined plugins that can be used to work with JavaScript tasks and on static content.

We will use these tools in frontend development because they are really helpful for specifying the project dependencies such that whenever the project needs to be ran on a different computer, with the command `npm install`, all dependencies are immediately in place, as they were previously saved into the `package.json` and `bower.json` files.

4.4.12. WebSockets

WebSockets represent an advanced technology used for opening an interactive communication session between two parts: the client and the server components. This technology is based on *WebSocket* - a computer communications protocol, which provides full-duplex communication due to a single TCP connection.

In order to establish a WebSocket connection, a protocol known as “handshake” is implemented: the client sends a WebSocket handshake request to which the server returns a WebSocket handshake response, as shown in the figure 4.14. This mechanism resembles HTTP in terms of allowing servers to handle HTTP connections as well as WebSocket connections on the same port.

Spring WebSockets will be used on the server side, since Spring Framework already integrates this module, whereas Stomp Over WebSocket messaging protocol is implemented on the client side.

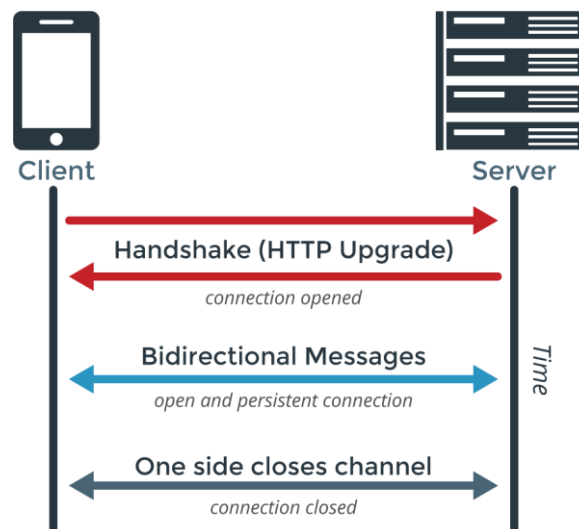


Figure 4.14 WebSockets communication flow

4.4.13. Google Geocoding API

Geocoding is the process of converting addresses (like a street address) into geographic coordinates (latitude and longitude), that one can use to place markers on a map, or position the map.⁴²

Reverse geocoding is the inverse process to geocoding, used for converting geographic coordinates into a human-readable address. The figure 4.15 exemplifies the structure of a geocoding request, as presented in the official documentation⁴³.

`http://maps.googleapis.com/maps/api/geocode/outputFormat?parameters`

Figure 4.15 Geocoding API request format

The *outputFormat* parameter indicates the type of the result (JSON or XML).

In the proposed application, the geocoding API will be used in order to obtain the latitude and longitude corresponding to addresses provided by the users. The JSON format is used for mapping the response and the location's latitude and longitude are extracted from it. An indicator will be placed on the map, based on the identified coordinates. For example, if the obtained latitude is 46.7712° and the longitude is 23.6236° , Cluj-Napoca would be indicated on the map as in figure 4.16.



Figure 4.16 Google Maps with pin-point on Cluj-Napoca

4.4.14. Git, Bitbucket and SourceTree

Git⁴⁴ represents an open source distributed version control system, which can be used to track changes in computer files and also to coordinate work on those files among team members.

⁴² <https://developers.google.com/maps/documentation/geocoding/start>

⁴³ <https://developers.google.com/maps/documentation/geocoding/intro>

A history of all the versions of the files within a project is kept, which provides the possibility of reverting the code to previous versions if necessary.

Bitbucket⁴⁵ is a web-based version control repository hosting service owned by Atlassian, for source code and development projects that use either Mercurial or Git revision control systems. Bitbucket offers free accounts with unlimited numbers of repositories, therefore it represents a good solution for storing the project. Some of its main features are pull requests with code review and comments, merge checks, documentation and issue tracking, which are all useful in the development process.

SourceTree is a free Git client for Windows and Mac, which provides an interface which replaces the command line, simplifying the use of Git.

The above presented technologies and tools simplify the code development process. The implemented code will be safely stored in a Bitbucket repository, where changes can be constantly pushed through commits, done directly from SourceTree. The possibility to rollback to previous commits offers the freedom of experimenting without affecting the proj

⁴⁴ <https://git-scm.com/>

⁴⁵ <https://bitbucket.org/>

Chapter 5. Detailed Design and Implementation

This chapter presents step by step the decisions taken during the development of this project, from design to implementation. The proposed system is composed of 3 subsystems: web application (front-end component), backend component and database.

An overview of the system will be offered through the client and server architecture diagrams, since the architectural pattern used is client-server, as presented in the section 4.1.1. The front-end component (client) contains the user interface and all the necessary logic to forward user requests to the back-end component (server), which is responsible for handling those requests, by using the business logic. Moreover, the main smaller modules of each subsystem will be described.

5.1. Web application architecture

5.1.1. General description

AngularJS is the framework used for developing the front-end component of the application and it is based on the **MVC** pattern. The view is defined in HTML, while the model and the controller are implemented in JavaScript.

Model View Controller (MVC) represents an architectural pattern, in which different aspects of the application are broken into components to separate responsibilities. The **Model** contains the data and logic, the **View** contains the visual layout and presentation, while the **Controller** connects the two.

Using this pattern, the various components have a relationship defined by their interactions. The model is only aware of itself. Generally, the view is aware of the model, as it's responsible for rendering data contained in the model and invoking actions (methods) on the model. The job of the controller is to create and populate the model and hand it over to the view. The controller needs to know about the model, and how to resolve the view and provide it the model.

In our project, the **AngularJS MVC** pattern is implemented using AngularJS, a JavaScript framework (described in section 4.4.10), as well as other technologies such as HTML, CSS, Bootstrap, already presented in the subchapter 4.2.

5.1.2. Components description

The figure 5.1 shows in details how different components are connected in the developed application. Starting from the **module**, which represents the container for the different parts of the application, to **directives**, which extend the power of HTML, each piece has its own responsibilities, which will be presented next.

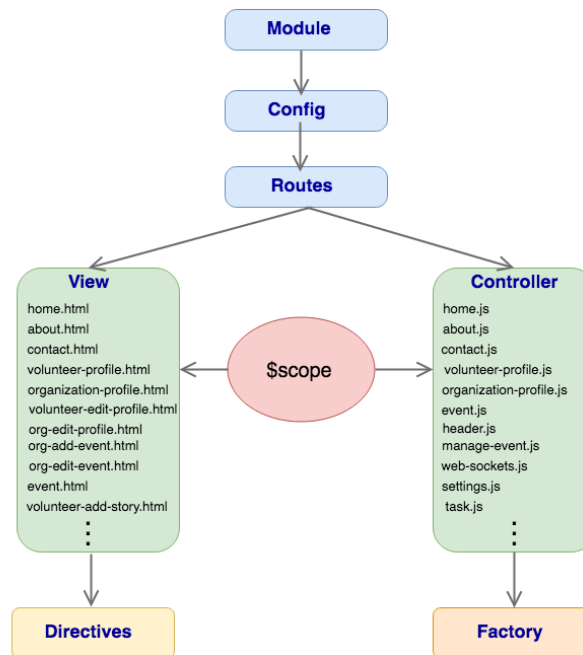


Figure 5.1 Detailed architecture of front-end component

5.1.2.1. Routes and View

The view in an AngularJS application is created with HTML. Each view is managed by Angular Router, which is used for defining the navigation paths within the application, based on the current location of the user. The figure 5.2 exemplifies how the routing is done in our application, with the following elements: *templateUrl*, which identifies the path to a certain view and *controller* together with *controllerAs*, referring to the associated controller of that view, responsible with handling the displayed data.

AngularJS is based on the Single Page Application (SPA) concept, which means that all the code (JS, HTML, CSS) is retrieved with a single page load. Moreover, the navigation between pages is performed without refreshing the whole page. Here comes the router's responsibility, by dynamically rendering a view, based on URL changes.

```

.config(function ($routeProvider, $locationProvider) {
  $routeProvider
    .when('/volunteer-profile', {
      templateUrl: 'views/volunteer-profile.html',
      controller: 'VolunteerProfileCtrl',
      controllerAs: 'volunteerProfileCtrl'
    })
    .when('/volunteer-profile/:volunteerId', {
      templateUrl: 'views/volunteer-profile.html',
      controller: 'VolunteerProfileCtrl',
      controllerAs: 'volunteerProfileCtrl'
    })
    .when('/organization-profile', {
      templateUrl: 'views/organization-profile.html',
      controller: 'OrganizationProfileCtrl',
      controllerAs: 'organizationProfileCtrl'
    })
})

```

Figure 5.2 AngularUI routing

The figure 5.3 presents how the main navigation pages, which can be accessed by all users, are connected, while the figure 5.4 shows only the pages which can be accessed by authenticated users, with minor differences. The page which can be used only by the organizations is manage-event page.

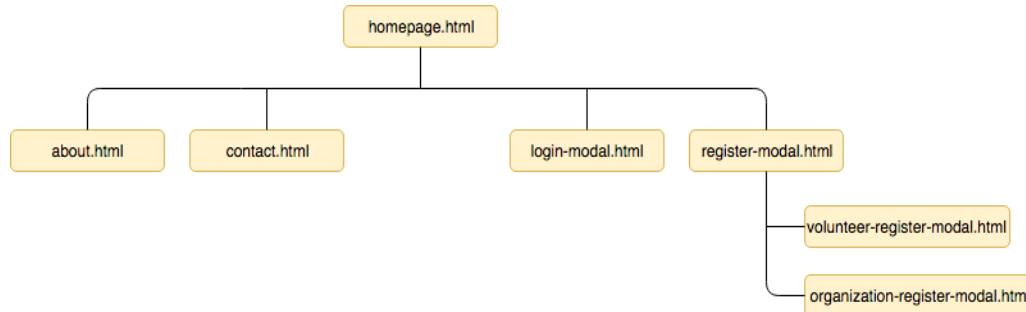


Figure 5.3 Guest's navigation diagram

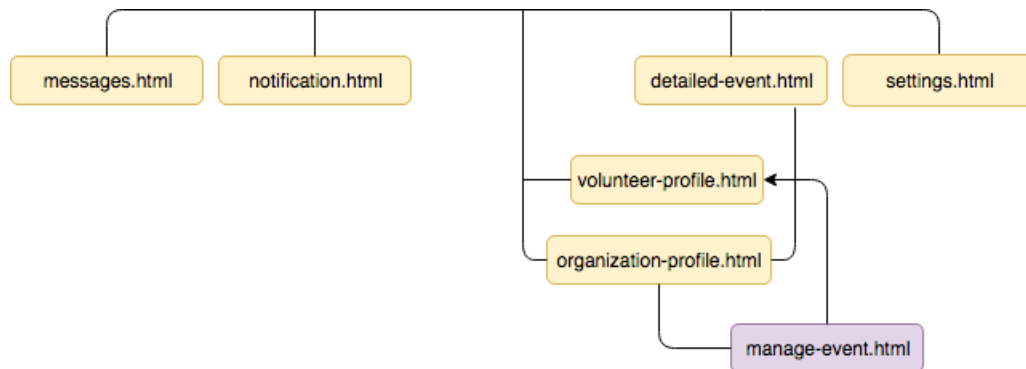


Figure 5.4 Authenticated user's navigation diagram

5.1.2.2. Controller

Controllers are one of the key components in the MVC architecture of AngularJS, used to control the data flow in the application. A controller is a JavaScript object which contains attributes/properties and functions and accepts different parameters.

The figure 5.5 shows how the event controller is implemented within our application, all the other controllers used being declared in a similar way. Before creating a controller, the **module** needs to be created, because it encapsulates all application components. The controller's name "EventCtrl" is followed by a list of parameters, from which "eventService" and "UserService" are services injected through dependency injection.

```

var volunteerApp = angular.module('volunteerApp');

volunteerApp.controller('EventCtrl', ['$scope', '$routeParams', '$location', 'eventService', 'UserService',
function ($scope, $routeParams, $location, eventService, UserService) {

```

Figure 5.5 Event Controller definition

In order to directly update the model, controllers and views are connected through *\$scope* object, which is the binding part between HTML and JavaScript. For populating *\$scope*, functions need to be defined and used in the controllers. Several controllers have been used: *HeaderController*, *HomeController*, *EventController*, *VolunteerController*, *OrganizationController*, *AdminController*, *TaskController*, *SocketController*, etc.

5.1.2.3. Services

Angular Services have the responsibility to provide application data/business logic to components. The service decides whether to provide mock data or go to server and fetch data from database/file/another service(s) etc. In this application, the services are actually communicating with the backend server through HTTP requests, in order to obtain the required data and forward it to controllers.

AngularJS services can be defined using the **factory** keyword or the **service** keyword. There are several differences between these two types of components, but the main one is related to the flexibility they offer:

- **Factory** is a function that returns the object directly, so basically what is needed to directly receive data in the controller, where the service function is called. The figure 5.6 is a template used for instantiating the majority of services within our application.

```
angular.module('volunteerApp')
  .factory('eventService', function ($http) {
    return {
```

Figure 5.6 EventService definition

- **Service** is a constructor functions of the object, which is instantiated with the **new** keyword and does not return anything explicitly. There is a particular situation when Service is used in the application, for creating an object holding the authenticated user details, which has to be common to all controllers. The figure 5.7 shows how the *UserService* is implemented, in order to identify the user's role and initialize the corresponding authenticated user instance: *OrganizationProfile*, *VolunteerProfile* or *Admin*.

```
angular.module('volunteerApp')
  .service('UserService', function (OrganizationProfile, VolunteerProfile, Admin) {

    console.log(OrganizationProfile.get(), VolunteerProfile.get(), Admin.get());

    this.user = {
      role: 'guest'
    };

    this.init = function(role, userData) {
      var userRole = role ? role.toLowerCase() : '';
      switch (userRole) {
        case 'organization': updateUser(OrganizationProfile.init(userData)); return OrganizationProfile; break;
        case 'volunteer': updateUser(VolunteerProfile.init(userData)); return VolunteerProfile; break;
        case 'admin': updateUser(Admin.init(userData)); return Admin; break;
        default: updateUser({role: 'guest'}); localStorage.clear(); return this.user
      }
    };

    this.getUser = function () {
      return this.user;
    };
  });
```

Figure 5.7 UserService definition

The factories used are: *volunteerService*, *organizationService*, *adminService*, *eventService*, *taskService*, *storyService*, *socketService*. The services used are: *UserService*, *VolunteerService*, *OrganizationService* and *AdminService*, used for authenticating the user and storing its credentials at the application level.

5.2. Backend Architecture

5.2.1. General description

For developing the back-end application, the main technology used is Spring Framework, as well as several Spring projects, as presented in the section 4.4.2.

The **Layered Architecture** is used for organizing the project structure, since it separates the code into four main categories: **presentation**, **application (business)**, **domain** and **persistence**, therefore providing **simplicity**, **consistency** and **browsability**, as shown in the figure 5.8.

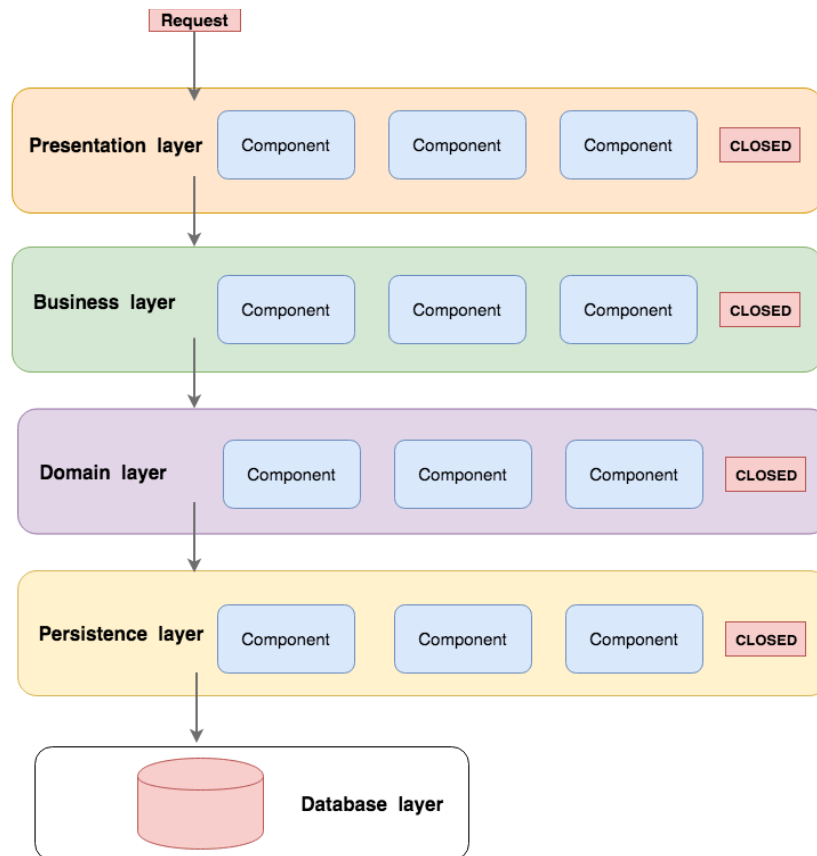


Figure 5.8 Layered architecture pattern

Next, each layer's responsibility will be described, explaining how it is mapped on our implementation needs:

- The **presentation layer** is usually responsible for presenting the application's UI to the end-client or managing client's requests. In our case, this layer is actually composed of controllers, which have the role of handling user requests, by using the services dependencies. The controllers forward data to the front-end component, which displays it.
- The **application layer (business logic layer)** contains all the logic that is required by the application to meet its functional requirements and, at the same time, is not a part of the domain rules. The services of our application are located at this level, being directly accessed by the controllers from the above layer. The application layer components work with models and repositories provided by the layers located below it.
- **The domain layer** represents the underlying domain that mostly contains domain entities and, in some specific cases, services. Business rules such as algorithms and invariants should all be a part of the domain layer. The models of this application compose the domain layer.
- The **persistence layer (infrastructure layer)** is responsible for persisting the data into the database and, in our case, consists of repositories through which **CRUD** (Create, Read, Update, Delete) operations are performed on the database.

The **database layer** is not part of this architectural pattern, but its presence in the figure 5.6 clarifies how the connection between the back-end server component and the database is realized through the **persistence layer**.

There are two important rules which were followed in order to correctly implement the Layered Architecture:

1. All the dependencies go in one direction, from presentation to infrastructure.
2. No logic related to one layer's concern are placed in another layer. This rule promotes two of the OO Design Principles, high cohesion and low coupling.

5.2.2. Components description

In the figure 5.9 the components of the application and their relationships are presented at a high level. The main components (packages) of the back-end application are: *Controller*, *Service*, *Repository* and *Model*. Some auxiliary components such as *Security*, *Config*, *Filter* and *Util* have been used. Next, the main purpose of each package will be described:

- **Controller** - contains a part of the application logic and has the role of forwarding user requests to services, as well as returning the required data provided by services as response to user request
- **Service** - encapsulates a major part of the business logic, being a middleware between controller and repository
- **Repository** - provides access to the database, using the data model
- **Model** - defines the data model of the application; the models represent entities which are persisted to the database
- **Security** - contains the JWT security related business logic
- **Config** - contains the WebSockets configuration

- **Filter** - communicates with the services and the models in order to filter the access to application resources
- **Util** - provides utility services such as mail service, http client service, etc. which are just “helping” the main services to accomplish their work

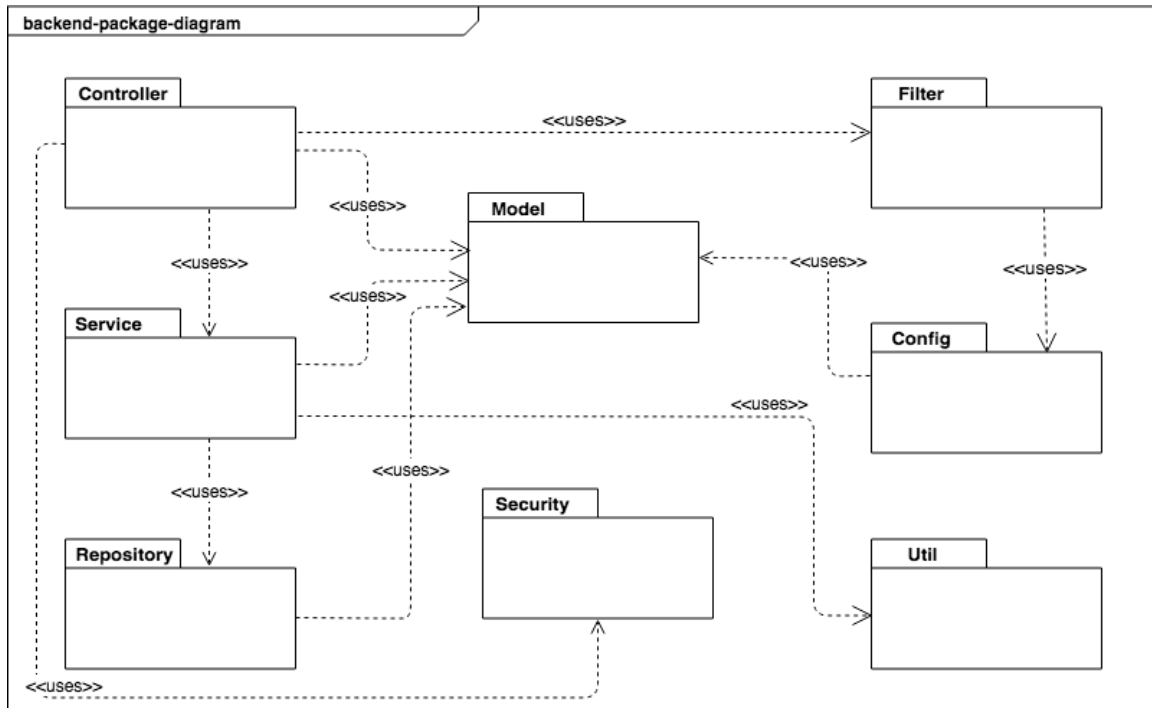


Figure 5.9 Backend package diagram

Next, the above presented packages and their relationships will be described in details.

5.2.2.1. Controller

This component is represented by several Java classes, which expose the application’s data through RESTful APIs. The controllers are located at the Presentation Layer and define the access points from external clients such as the front-end component. When a request from the front-end application is intercepted, the corresponding controller handles it and delegates part of the responsibility to the linked service, which executes the request. Afterwards, an HTTP response is sent back to the front-end.

Each Java class from the Controller package uses several annotations, from which the most used will be presented:

- **@RestController** - the DispatcherServlet provides a single entry point in the application and it delegates the requests to additional components such as actual controller classes, which are annotated with *@RestController*. So these controllers have the role of handling web requests. This annotation replaces *@Controller* and *@ResponseBody*.

- **@GetMapping** - it is a specialized version of *@RequestMapping* annotation, that acts like a shortcut for *@RequestMapping(method = RequestMethod.GET)*. *@GetMapping* annotated methods handle the **HTTP GET** requests matched with given **URI** expression.
- **@PostMapping, @PutMapping, @DeleteMapping** - similar to *@GetMapping*, the only difference is that the corresponding annotated methods handle HTTP POST, HTTP PUT and HTTP DELETE request, respectively.
- **@Autowired** - is used for automatic dependency injection. The annotation is used for injecting services into controllers and repositories into services, as well. The dependency goes always in the direction **controller -> service -> repository**, which means that controllers depend on services, while services depend on repositories, but the inverse dependency is not allowed (layered architecture).
- **@PathVariable, @RequestParam, @RequestBody** - define the types of parameters which can be used to perform an HTTP request, basically to make a request to a web service.

In the figure 5.10 it is shown how the previous presented concepts are applied on a controller class in our application. The definition of a RESTful web service which retrieves the task having the specified identifier *id* is shown.

```
@RestController
public class TaskController {

    @Autowired
    private ITaskService taskService;

    @GetMapping("task/{id}")
    public ResponseEntity<?> getTaskById(@PathVariable int id) {
        Task task = taskService.findById(id);
        if (task != null) {
            TaskDTO taskDTO = taskService.mapTask(task);
            return new ResponseEntity<>(taskDTO, HttpStatus.OK);
        }
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
}
```

Figure 5.10 Definition of TaskController

Other controllers which are defined in a similar way are: *AdminController*, *VolunteerController*, *OrganizationController*, *EventController*, *StoryController*, *WebSocketsController*, each consisting of several REST endpoints which expose the application's data.

5.2.2.2. Service

This part of the system architecture is responsible with the business logic. As presented before, the service component represents the middleware between **controller** and **repository**, which means that it receives requests from the higher level (controller) and it utilizes the data model and the database access level, in order to execute those requests.

Java classes within Service package are marked as Spring Beans through the **@Service** annotation, denoting their role of performing service tasks. Each service class implements a corresponding interface. Direct communication with the lower level - data access layer - is possible due to the **@Autowired** annotation presented before, which marks this time the injected repositories. The figure 5.11 shows how a service class is declared in our application and how the service method which retrieves the comments corresponding to an event is implemented.

```
@Service
public class EventService implements IEventService {

    @Autowired
    private EventRepository eventRepository;

    @Override
    public List<EventCommentDTO> getCommentsForEvent(int id) {
        Event event = eventRepository.findOne(id);
        List<EventComment> comments = new ArrayList<>();
        if (event != null) {
            comments = event.getComments();
        }
        return mapEventComments(comments);
    }
}
```

Figure 5.11 Definition of EventService

The other services used are organized in a similar manner, in order to maintain the high cohesion between classes. The following services have been implemented: *UserService*, *VolunteerProfileService*, *OrganizationProfileService*, *EventService*, *TaskService*, *VolunteerToEventService*, *LocationService*, *MessageService*, *EventCommentService*, *VolunteerTaskService*, *VolunteerFeedbackService*, together with the associated interfaces.

5.2.2.3. Repository

The Data Access Layer communicates with the Data Storage Layer to perform CRUD operations. CRUD represents an acronym for the database operations Create, Read, Update, and Delete. By extending the **JpaRepository** presented in the section 4.4.4, these operations are inherited, so there is no need to actually write their implementation. This represents an advantage since the developer does not need to focus on how to code repetitive methods, being able to just use them.

In order to use this facility, the interfaces are annotated with **@Repository** and they have to extend the predefined interface **JpaRepository<ID, T>**, where ID represents the identifier's type, while T is the class type of the entity for which the repository is created.

If more complicated operations are required, a template can be used. You just have to declare the method and the repository will know how to translate it. As exemplified in the figure 5.12, a method for finding a volunteer profile with the specified first name and last name would be:

```
@Repository
public interface VolunteerProfileRepository extends JpaRepository<VoterProfile, Integer> {
    VoterProfile findByFirstNameAndLastName(String firstName, String lastName);
}
```

Figure 5.0.12 Implementation of Repository interface for VoterProfile entity

We will not present all the created repositories, since for each entity there is a corresponding repository through which it is created, read, updated and deleted from the database.

5.2.2.4. Model

This section presents the application's data model. The structure of each entity will be specified, together with its role. Besides the entities which are persisted to the database, another type of models called DTOs and having similar structure as the entities are used, in order to transfer data from and to the database.

The implemented Java classes are:

- **User** - defines the user related data, common to all user types (Voter, Organization, Administrator), containing also properties specific to user account; the username, e-mail, password, user role, and also the account related information.
- **VoterProfile** - contains information provided by the voter user at registration, such as first name, last name, date of birth, address, skills, interests, experience, etc. which is used for creating a personal profile.
- **OrganizationProfile** - stores organization specific data: name, acronym, date of creation, description, website and facebook links, as well as information about the actual person who manages the organization's account
- **Event** - is an activity presented by an organization and described through the following attributes: title, description, location, hour interval, category, total number of volunteers needed, organizer.
- **EventCategory** - defines the category of an event; an event can be associated to several categories
- **EventComment** - describes the attributes of a comment to an event; it identifies the user who added the comment, the date when it was posted and the text of the comment.
- **Location** - represents the place where the events are organized and contains the address, as well as the corresponding longitude and latitude, which are used for placing a marker on Google Maps
- **Task** - represents 'a piece of work' which needs to be done by volunteers in the context of an event, and it contains information in which both volunteers and organizations are interested: title, description, date

- (because it might differ from the event's actual date), location, hour interval and status.
- **Story** - is created by a volunteer in order to share thoughts about an event with other users and it contains a title, a short description and a link to that event.
- **VolunteerToEvent** - used for storing the status of a volunteer to an event, which can be one of: *SAVED*, *APPLIED*, *ACCEPTED*, *REJECTED*, *LEFT*; since many volunteers have different statuses to many events, a connecting entity was required.
- **VolunteerFeedback** - represents the characteristics of the feedback offered by the organization to each volunteer, regarding their engagement to events; a description and a relevant title should be added
- **Message** - defines the attributes of the messages exchanged between the users of the application: sender, receiver, time of creation, actual message content, a flag indicating if it is read or not.

The figure 5.13 shows how the model, repository, service, service interface and controller classes are connected, in order to access, manipulate and transmit event's data. These relationships are valid for all the other models and their associated classes. Moreover, an UML class diagram presenting the most important relationships can be consulted in the Appendix 4, at the end of the paper.

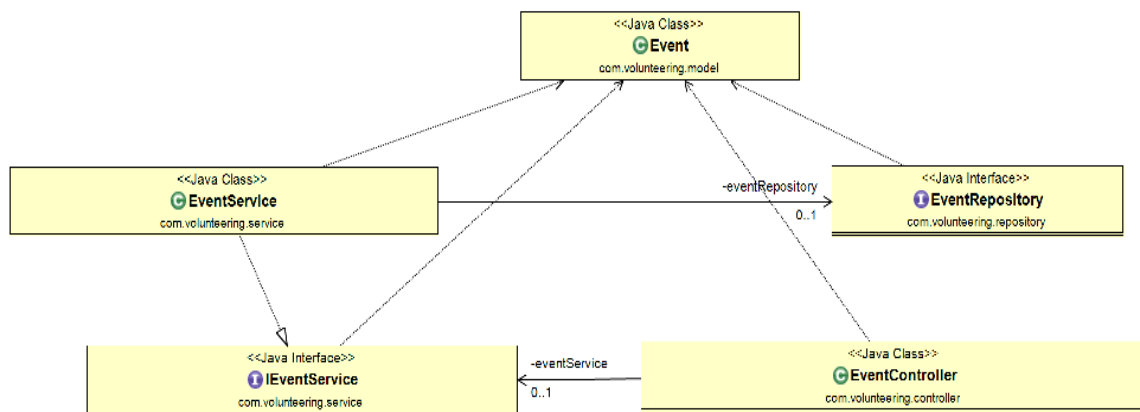


Figure 5.13 Classes interaction

5.2.2.5. Security

The security system is implemented using JWT (JSON Web Tokens). JWT is used for verifying the user's identity at login and each request done after the user is authenticated includes the generated token. The figure 5.14 shows how the user's role is verified at login, when the unique token is generated.

```

List<String> roles = new ArrayList<>();
roles.add(user.getUserRole().toString());
String token = Jwts.builder().setSubject(user.getUsername()).claim( s: "roles", roles).setIssuedAt(new Date())
    .signWith(SignatureAlgorithm.HS256, s: "secretkey").compact();
  
```

Figure 5.14 JWT check of user's role

5.2.2.6. Config

This package contains the WebSocket configurations, which enable the communication between the client and server. In the developed application, this protocol is used in order to provide an efficient way of sending and receiving messages between users, as well as for providing instant notifications whenever a volunteer is accepted/rejected to an event.

5.2.2.7. Filter

The navigation filters of the application are defined at this level. The filter has the role of intercepting any external request and of validating it. Two filters have been used. The first one is the *Cors Filter*, which specifies the access configurations: the origin of the request, the HTTP methods accepted, the maximum duration, as well as the header types allowed. The figure 5.15 presents the method used for filtering the incoming requests.

```
@Override
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
    throws IOException, ServletException {
    HttpServletResponse response = (HttpServletResponse) res;
    HttpServletRequest request = (HttpServletRequest) req;

    response.setHeader("Access-Control-Allow-Origin", s1: "http://localhost:9000");
    response.setHeader("Access-Control-Allow-Methods", s1: "POST,PUT,GET,OPTIONS,DELETE");
    response.setHeader("Access-Control-Max-Age", s1: "3600");
    response.setHeader("Access-Control-Allow-Headers", s1: "X-Requested-With, X-Auth-Token, Content-Type, Authorization");
    response.setHeader("Access-Control-Allow-Credentials", s1: "true");

    if (!"OPTIONS".equalsIgnoreCase(request.getMethod())) {
        chain.doFilter(req, res);
    }
}
```

Figure 5.15 Cors Filter implementation

The second filter is the JWT Filter, which is responsible for blocking unauthorized requests, if the user who initiates them does not possess a valid token.

5.2.2.8. Util

The Util package contains services such as *MailService*, *HttpService* and *DozerService* which are used by the main services. The MailService is implemented using the JavaMail API, in order to send different types of emails to registered users. The HttpService is used for making a request to the geocoding API provided by Google, to obtain the coordinates of a certain address.

A special type of service used is *DozerService*, which is implemented using the singleton design pattern. Dozer is a Java Bean to Java Bean mapper that recursively copies data from one object to another, attribute by attribute. This service provides the Data Transfer Objects (DTO) from which the response to a certain request is built. The figure 5.16 shows how the Dozer singleton mapper is used to convert the entity User to the data transfer object UserDTO.

```
@Override
public UserDTO mapUser(User user) {
    DozerBeanMapper mapper = dozerService.getDozer();
    return mapper.map(user, UserDTO.class);
}
```

Figure 5.16 User object mapped with DozerBeanMapper

5.3. Database

In order to store the application data, MySQL database is used, since it is the world's most popular open source database and it presents several advantages emphasized in the section 4.2.6. Spring provides specific annotations for mapping the relationships between entities which represent in fact the relationships between database tables. Such annotations are: *@OneToOne*, *@OneToMany*, *@ManyToMany*, etc.

The figure 5.17 presents the database diagram, which is composed of 14 tables and the relationships between them. Since MySQL is a relational database, each table is linked to at least one other table.

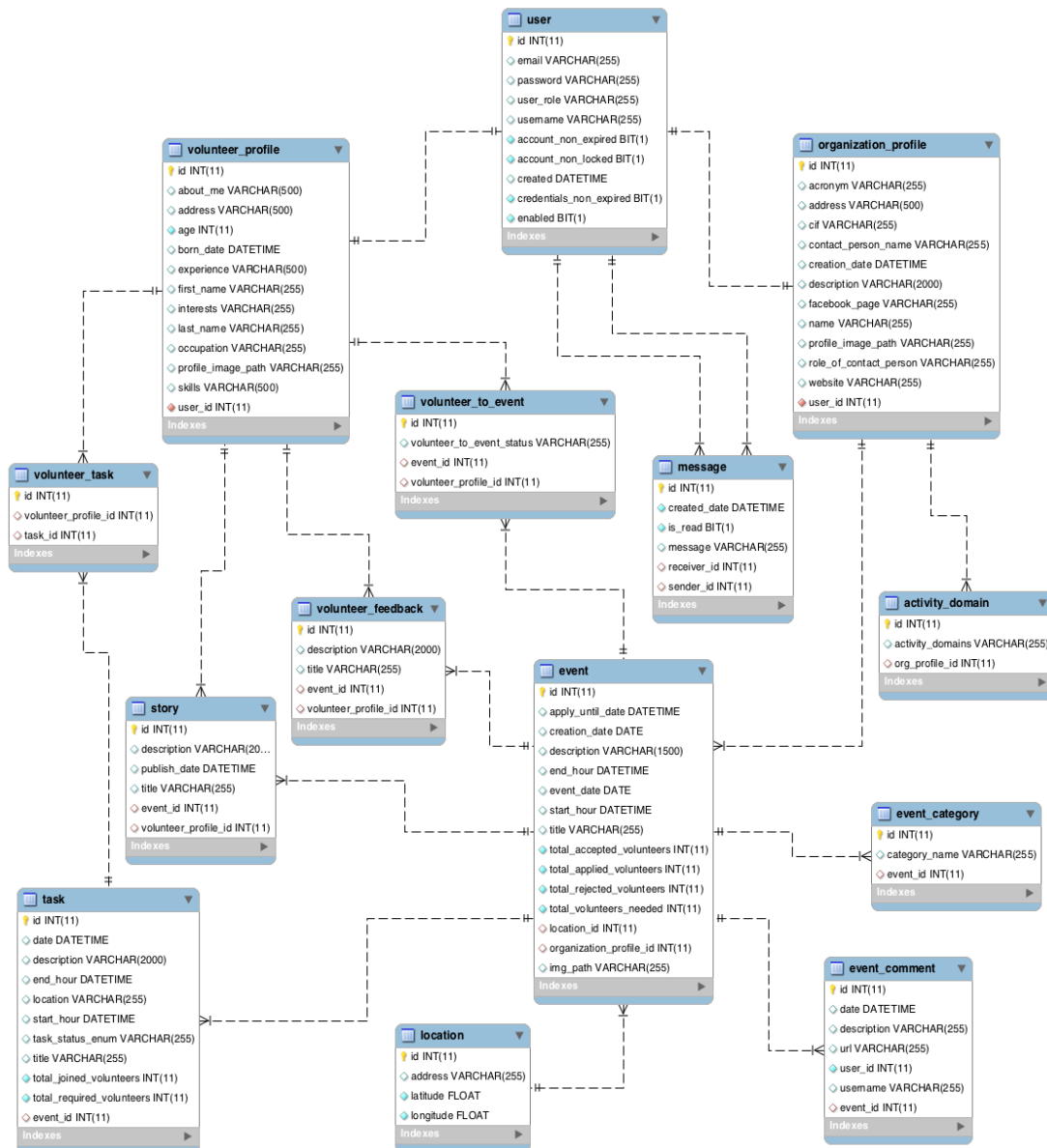


Figure 5.17 Database diagram

Next, we will present the main tables of the database, their connection to other tables and the contained fields, together with their data types.

User table contains the authentication information which is common for all system's users: the credentials used for accessing the application, as well as account specific data. The following fields are used:

- **id:** primary key of the table; type: int
- **email:** stores the user's email address; type: varchar
- **username:** unique identifier, part of user's credentials; type: varchar
- **password:** secret token used for authentication, part of user's credentials; type: varchar
- **user_role:** identifies the user type and can be volunteer, organization or admin; type: varchar
- **account_non_expired:** flag to indicate if the account is expired; type: boolean
- **account_non_locked:** flag to indicate if the account is locked by the admin; type: boolean
- **credentials_non_expired:** flag which indicates if the users credentials are expired; type: boolean
- **created:** account creation date; type: datetime

Volunteer_profile table stores the personal data of the volunteer user, provided at registration or by using the profile editing functionality. The table is composed of the next columns:

- **id:** primary key, matching the user's primary key; type: int
- **first_name:** user's first name; type: varchar
- **last_name:** user's last name; type: varchar
- **born_date:** user's date of birth; type: datetime
- **age:** age calculated based on the provided date of birth; type: int
- **address:** user's address; type: varchar
- **phone:** personal phone number; type: varchar
- **about_me, experience, skills, interests:** short description provided by the volunteer; type: varchar
- **occupation:** job/career information; type: varchar

This table is connected to **volunteer_task**, **volunteer_feedback**, **volunteer_to_event** and **story** tables.

Organization_profile table contains each organization's data, which is specific to this type of user. The following information is stored:

- **id:** primary key of the table; type: int
- **name:** organization's name; type: varchar
- **acronym:** abbreviation of organization's name; type: varchar
- **description:** general information about the organization; type: varchar
- **address:** organization's headquarters or office location; type: varchar
- **cif:** unique code by which an organization is identified; type: varchar
- **creation_date:** date when the organization's activity started; type: datetime

- **contact_person_name:** name of the person who uses the account for the organization; type: varchar
- **role_of_contact_person:** role of the organisation's representative; type: varchar
- **website, facebook_page:** links to the website and facebook page; type: varchar

This table is connected to **activity_domain** and **event** tables.

Event table holds the event related information and we could say that is the 'central' table of the database, since it is connected to many other tables. The contained fields are:

- **id:** primary key of the table; type: int
- **title:** identifies the title of the event; type: varchar
- **creation_date:** date when the event is created; type: date
- **event_date:** date when the event takes place; type: date
- **apply_until_date:** deadline for applying as volunteer to event; type: date
- **description:** short summary of the event purpose to attract volunteers; type: varchar
- **total_needed_volunteers, total_applied_volunteers, total_accepted_volunteers, total_rejected_volunteers:** the number of volunteers needed, who applied, who are accepted or rejected, respectively; type: int
- **start_hour, end_hour:** hour interval of the event; type: datetime
- **location_id:** foreign key, indicating the event's location; type: int
- **organization_profile_id:** foreign key, representing the identifier of the organizer

Task table stores all the tasks created for the events. It is directly connected to the event table and indirectly (through **volunteer_task**) to the **volunteer_profile** table. The table is composed of the following columns:

- **id:** primary key of the table; type: int
- **title:** identifies the title of the task; type: varchar
- **description:** presents in details the volunteers' responsibility; type: varchar
- **date:** date when task should be done; type: date
- **start_hour, end_hour:** hour interval of the task; type: datetime
- **task_status:** can be to_do, in_progress, done or canceled; type: varchar
- **location:** indicates the task's location, because it might differ from the event's location; type: varchar
- **total_required_volunteers, total_applied_volunteers:** number of volunteers needed to accomplish the task and the number of volunteers who already applied; type: int
- **event_id:** foreign key, indicating the identifier of the event for which the task is created; type: int

This table is connected to **event** and **volunteer_task** table, which is actually the connection between **task** and **volunteer_profile** tables.

Story table contains the stories added by volunteers about the events they participated at. The following information is stored:

- **id:** primary key of the table; type: int

- **title:** identifies the title of the story; type: varchar
- **description:** summary of the volunteering experience; type: varchar
- **date:** date when the story is created; type: datetime
- **event_id:** foreign key, representing the identifier of the event to which the story is related; type: int
- **volunteer_profile_id:** foreign key, representing the identifier of the volunteer user who added the story; type: int

This table is linked to the **event** and **volunteer_profile** tables, since a story is added by a volunteer and it should be related to an event.

Chapter 6. Testing and Validation

This chapter presents the testing and validation methods used for checking if the system behaves as expected. Both manual and automated test tools have been used in order to cover as many functionalities as possible.

6.1. Manual testing

It represents the process of using the features and functions of the application as an end user and verify if the software is working as required. In order to successfully conduct manual tests, the software requirements must be completely understood by the tester. The next step is to write test cases, in order to ensure good test coverage. Since manual testing is necessary in order to ensure a good user experience and a high level of quality, this testing method was used in our case. We will present two of the test cases used for identifying bugs within the developed application: *Organization creates an event*, which is described in the table 6.1 and *Volunteer applies to event*, captured in the table 6.2.

6.1.1. Test case 1: Organization creates an event

Preconditions: User must have a valid organization account

Table 6.1 Steps of test case 1

No.	Action	Expected result
1	Open the web application	The homepage is displayed
2	Select <i>Login</i> option from the navigation bar	The Login modal window containing the username and password input fields is displayed
3	Enter required credentials and press the <i>Login</i> button	The user is redirected to the organization-profile page
4	Click on the <i>Add event</i> button	A modal containing the event's title, location, date, application deadline, description, hour interval, category and required no. of volunteers is displayed
5	Click on the <i>Add event</i> button	If the input is valid, a success message appears, otherwise an error message is displayed
6	Visualize created event	The created event is displayed on the organization's profile page, as well as on homepage

6.1.2. Test case 2: Volunteer applies to event

Preconditions: User must have a valid volunteer account

Table 6.2 Steps of test case 2

No.	Action	Expected result
1	Open the web application	The homepage is displayed
2	Select <i>Login</i> option from the navigation bar	The Login modal window containing the username and password input fields is displayed
3	Enter required credentials and press the <i>Login</i> button	The user is redirected to the volunteer-profile page
4	Access the homepage	The homepage is displayed
5	Search the event to which volunteer wants to apply	The searched event is displayed
6	Hover the event component	The <i>Save</i> and <i>Apply</i> buttons are visible now
7	Click on the <i>Apply</i> button	A success message appears if the user has not already applied to that event or if he was not rejected, otherwise an error message appears
8	View event to which the volunteer applied	The event should be displayed when selecting <i>Already Joined</i> on homepage or on the volunteer-profile page under the <i>I applied to</i> label

By following the steps described in the test cases, it can be noticed if the system behaves as expected when invalid input is provided. This can be done by checking if corresponding success and error messages are displayed as a result of user's actions.

6.2. Automation testing

While manual testing done by humans requires physical time and effort to ensure the software code works correctly, automation testing is done through an automation tool, less time is needed in exploratory tests and more time is needed in maintaining test scripts for increasing overall test coverage.

Some of the essential tools, frameworks, and libraries that can be used by developers for writing unit tests and integration tests when coding in Java are: **JUnit**, **REST Assured**, **Selenium**, **Postman**, **TestNG**, **Mockito**, **Cucumber** and **Robot Framework**.

The figure 6.1 presents the test automation pyramid, showing that the number of required tests decreases as advancing from unit tests (lowest level) to UI testing (highest level).

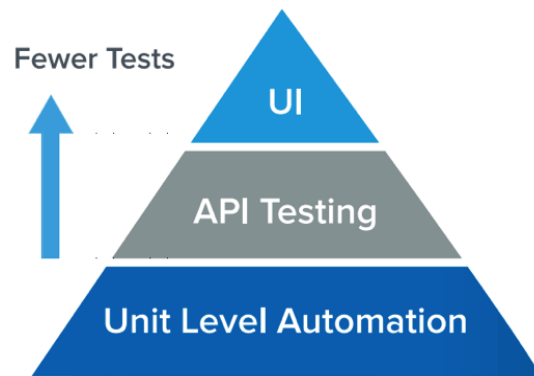


Figure 6.1 Test automation pyramid

Next, we will present the tools and frameworks used at each level of the test automation pyramid, for testing the developed web application. Some of them are part of the list presented before.

- **Unit Level Automation**

A *unit test* is a piece of code written by a developer that executes a specific functionality in the code to be tested and asserts a certain behavior or state.

JUnit⁴⁶ is a unit testing framework for Java Programming language and is used in test-driven development. Almost all major IDEs, e.g. Eclipse, NetBeans, and IntelliJ, provide JUnit integrations, which means you can both write and run the unit test right from those IDEs.

A *JUnit test* is a method contained in a class which is only used for testing. This is called a *Test class*. To define that a certain method is a test method, annotate it with the `@Test` annotation.

Since a unit test targets a small unit of code, e.g., a method or a class, we used it for testing simple methods.

- **API Testing**

API testing involves testing the collection of APIs and checking if they meet expectations for functionality, reliability, performance, and security and if the correct response is returned.

⁴⁶ <https://junit.org/junit4/>

The tool used for performing API testing is **Postman**⁴⁷, which is a Google Chrome app for interacting with HTTP APIs. It has a friendly GUI for constructing requests and reading responses.

The majority of our application's endpoints have been tested using Postman. The figure 6.2 shows how a request to ***http://localhost:8080/events*** is done and which is the obtained result. Since this REST API can be accessed without authentication, we just have to specify the HTTP method used (**GET**), the endpoint (previously specified) and select **NoAuth** type. For visualizing the result we chose JSON format. The HTTP status code obtained is **200** (OK).

- **UI Testing**

For testing the UI component of the application, the **DevTools**⁴⁸ provided by Chrome was used during the implementation. It is a set of web developer tools built directly into the Google Chrome browser, which can help you diagnose problems quickly, and help you build better websites, faster.

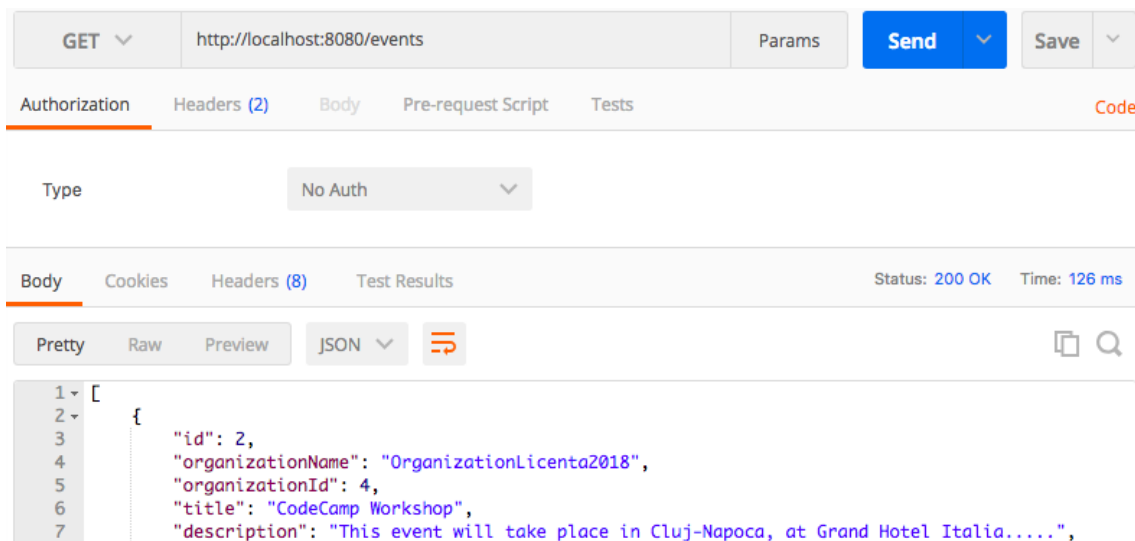


Figure 6.2. Example of API call using Postman

⁴⁷ <https://www.getpostman.com/>

⁴⁸ <https://developers.google.com/web/tools/chrome-devtools/>

Chapter 7. User's manual

This chapter presents the hardware and software resources required to configure and run the developed project on a computer. A set of instructions for installing each component of the system, as well as utilization instructions together with relevant pictures are provided. Therefore, new system users should be able to easily overcome the challenge of using the developed web application.

7.1. System requirements

In order to provide an optimal environment for running the application, the computer used should have the following specifications:

- **CPU:** Intel Core i5
- **Frequency:** 1.8 GHz
- **RAM memory:** 4 GB
- **Operating system:** Windows, Linux or OS X
- **Web browsers:** Google Chrome, Mozilla Firefox, Safari, Opera

7.2. Installation and running

The back-end and the front-end projects are both hosted on **Bitbucket**, into separate repositories, since they do not contain common code. Therefore, the first step would be to clone each repository on the local machine, then configure and install all the required dependencies of the back-end project, together with the database. Finally, the setup of the front-end project should be done.

Next, we will present step by step the phases described above.

Step 1: Obtain a local copy of the projects

- Download and install **GIT**⁴⁹
- Require access to both repositories since they are private, as shown in the figure 7.1
- Clone each Bitbucket repository in a separate empty local directory using the “**git clone <repository-url>**” command or directly from Bitbucket, using a Git client such as SourceTree or SmartGit

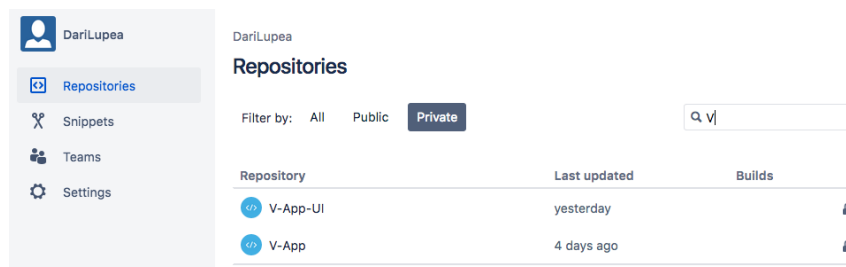



Figure 7.1 Bitbucket repositories overview

⁴⁹ <https://git-scm.com/>

Step 2: Setup back-end project

- Download and install **Java SE Development Kit 8**⁵⁰
- Download and install **IntelliJ IDEA**⁵¹ or other Java IDE (such as Eclipse, NetBeans, etc.)
- Download and install **Apache Maven 3.5.4**⁵²
- Import project in IntelliJ: *open IntelliJ -> Import Project -> Select directory containing the back-end project -> Next -> Import project from external model and select Maven -> Next -> Finish*. If another IDE is used, similar steps should be followed.
- Before running the project, the database should be created, as shown at **step 3** presented below.
- The **application.properties** file should specify the configuration for enabling the database connection with the user credentials, as exemplified in the figure 7.2.
- Run the application by clicking  or Right Click on *MyApplication.java* -> *Run MyApplication.main()*
- The application should be now available on port 8080, so go to any web browser and enter <http://localhost:8080/events> to see if any results are displayed

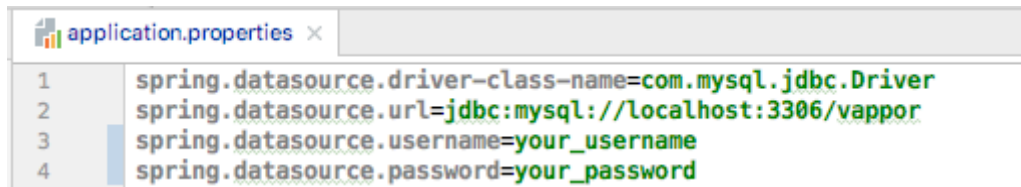


Figure 7.2 Database configuration in application.properties

Step 3: Setup the database

- Download and install **MySQL Installer**⁵³, which provides a wizard-based installation experience for all the MySQL software needs
- Start **MySQL Server**
- Open **MySQL Workbench** application and create a new connection and a new user
- Create a new schema called “**vappor**”
- Import the vappor.sql script containing the database (which can be found in the directory of the back-end project): *Data Import -> Import from Self-Contained file -> Select the vappor.sql script file -> Start import*
- Create the database and populate it by executing the imported script

Step 4: Setup front-end project

- Import project in IntelliJ, this time skipping the Maven configuration part

⁵⁰ <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

⁵¹ <https://www.jetbrains.com/idea/download/>

⁵² <https://maven.apache.org/download.cgi>

⁵³ <https://dev.mysql.com/downloads/installer/>

- In the file system, navigate to the directory where **package.json** file is located, open a terminal window and use the command **npm install**, for installing all the dependencies required by the project
- Run the application using the “**grunt serve**” command from the terminal window provided by IntelliJ
- The application starts on the port 9000: <http://localhost:9000/>

If the project is correctly configured at this point, the communication between the front-end (client) and back-end (server) components should be enabled and the user should be able to navigate the application, as presented in the following section.

7.3. Utilization instructions

A walk-through of the application will be done, by presenting each web page that the user can access, based on its role: guest, volunteer, organization or administrator. The sitemap of the web application will be shown, in order to help user gain a better visual perspective over the use case scenarios of the developed software system.

7.3.1. Sitemap

The sitemap presented in the figure 7.3 shows how different pages of the application are connected and which users are able to access them. The sitemap was very useful during the design phase, offering an overview of the user interface and it was modified during the implementation phase, based on required changes.

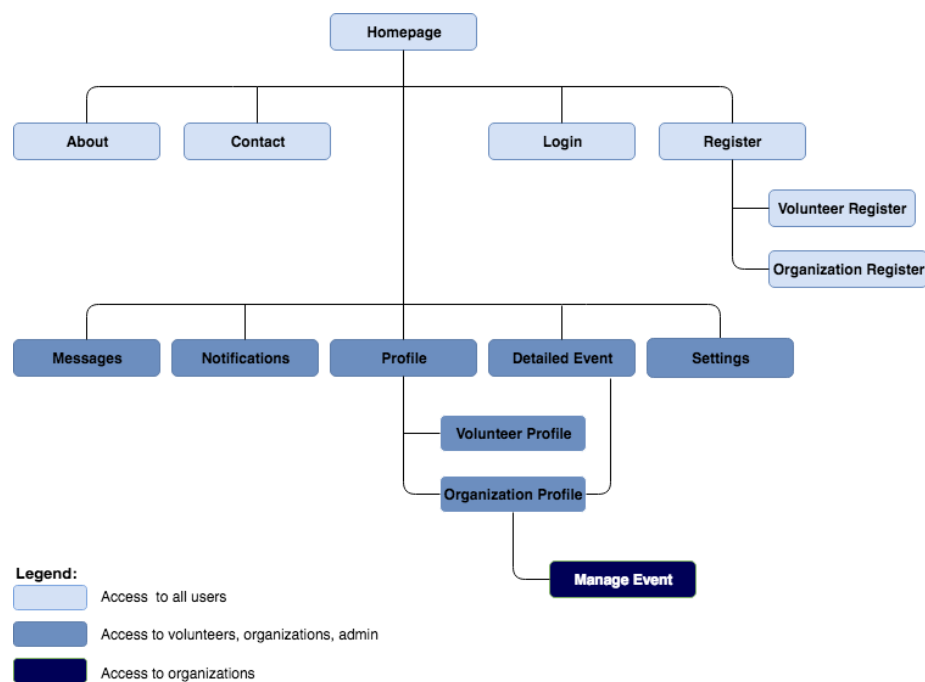


Figure 7.3 Sitemap

7.3.2. Application navigation

Each page of the application, together with a relevant image will be presented, starting with the homepage and other pages which can be accessed by all users and continuing with pages to which the access is restricted based on user's authorization. The main use cases will be emphasized along with the website navigation description.

- **Homepage**

The first page the user can interact with when entering the application is the homepage, presented in the figure 7.4. It can be accessed by any user, since it presents an overview of the events added by the organizations and provides mechanisms of searching events and of filtering by date (using the calendar component), by category or by status (coming soon or passed).

Other pages which can be directly accessed from homepage are **About**, **Contact** and **Detailed Event** pages.

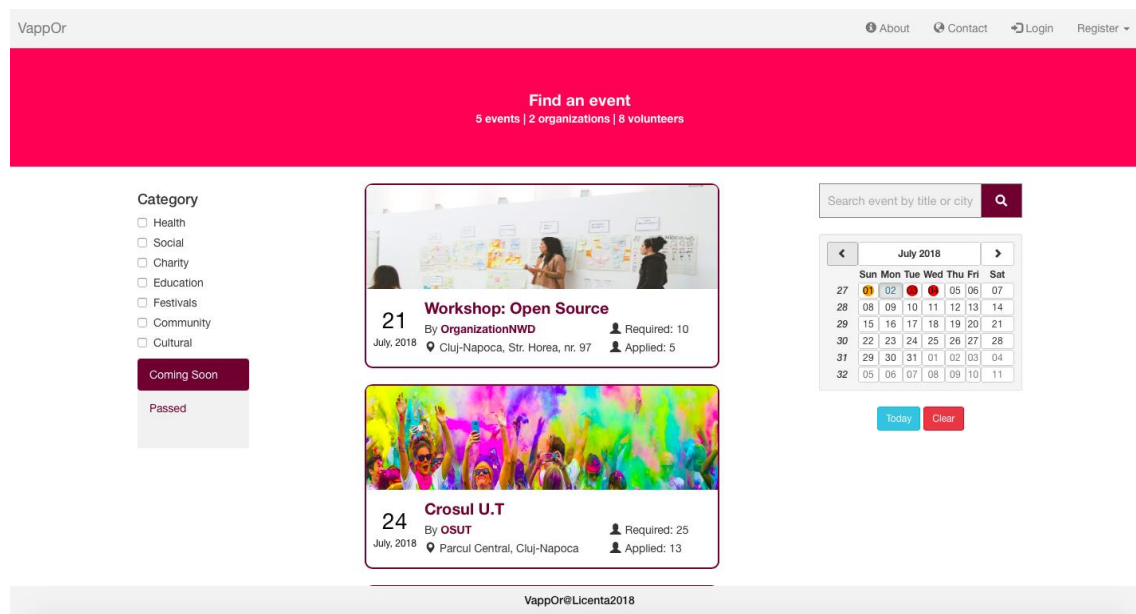


Figure 7.4 Homepage

The unauthenticated user is able to **Login** using a username and a password (figure 7.5) if he already has an account, otherwise he can **Register** as a **Volunteer** (figure 7.6) or as an **Organization** (figure 7.7), by providing the required information.

The image shows two overlapping modal windows. The 'Login' modal on the left has a title 'Login', fields for 'Username' and 'Password', a 'Remember me' checkbox, a 'Forgot password?' link, and 'OK' and 'Cancel' buttons. The 'Volunteer register' modal on the right has a title 'Volunteer register' and fields for 'Email', 'Username', 'Password', 'Confirm password', 'First name', 'Last name', 'Address', 'Occupation', 'Born date', and 'Phone'. It includes 'Register' and 'Cancel' buttons at the bottom right.

Figure 7.5 Login modal

Figure 7.6 Volunteer register modal

The image shows the 'Organization register' modal. It has a title 'Organization register' and fields for 'Email', 'Username', 'Organization's Name', 'Password', 'Confirm password', 'Acronym', 'CIF', 'Contact person's name', 'Role in organization', 'Address', 'Creation date', 'Website', and 'Facebook page'. At the bottom, there is a section for 'Activity domains' with checkboxes for Health, Social, Charity, Education, Festivals, Community, Cultural, and Art. 'Register' and 'Cancel' buttons are at the bottom right.

Figure 7.7 Organization register modal

- **Detailed Event Page**

Any user can access this page by clicking on the event's title or date from the media component containing the most important event details, which is displayed on homepage. The detailed event page has three main horizontal sections, as it can be noticed from the figure 7.8: the first one presents the title, date and organizer of the event (and a section which can be used by volunteers to save or apply to event), the second offers more details about the event such as the hour interval, the location marked also on Google maps, a representative image and a short description. The third section is represented by the comments area, which can be used by authenticated users to discuss about the event and can be just viewed by unauthenticated users.

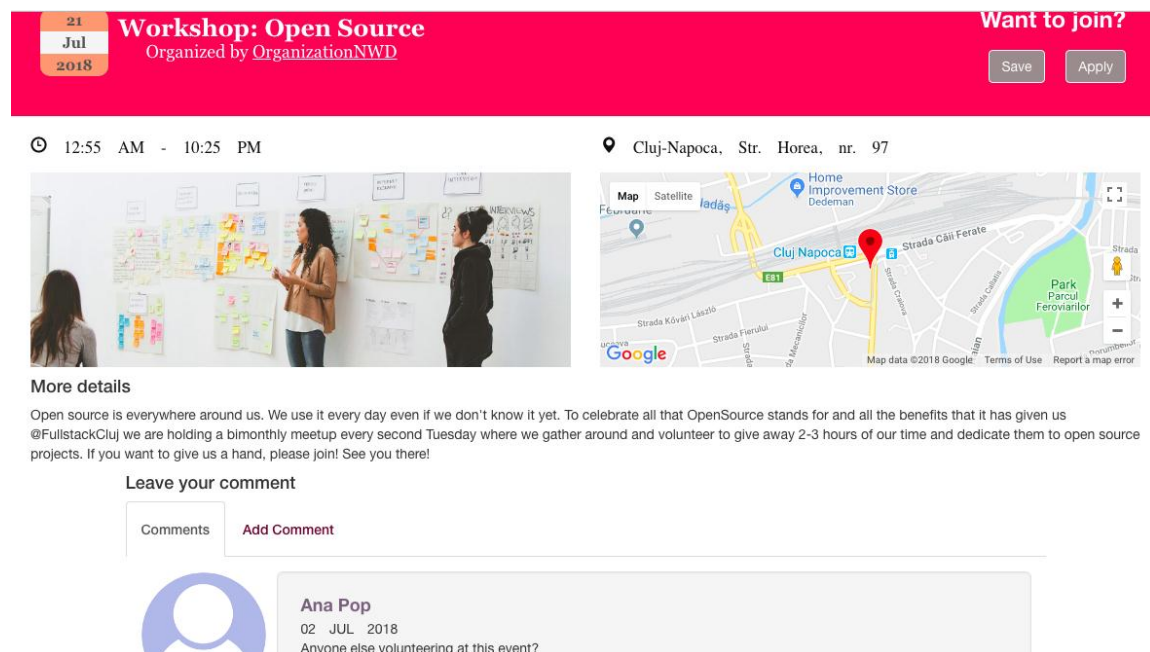


Figure 7.8 Detailed event page

- **Volunteer-Profile Page**

After authentication as a volunteer user, the volunteer-profile page is displayed (figure 7.9). It presents personal information about the volunteer, which is relevant in the volunteer selection process done by organizations. The volunteer is able to edit his profile, to view the saved events or the events he applied to. The status (accepted or rejected) to a certain event can also be checked. If tasks have been assigned to the volunteer, they are visible on this page. Moreover, he can add stories about the events he participated at, using the template from the figure 7.11, and visualize them. Other authenticated users (volunteers and organizations) can visit the volunteer-profile page, but will see only a part of this data.

The screenshot shows a volunteer profile for 'Dari Lupea', age 22. The profile includes a photo, an 'Edit profile' link, and sections for 'About me', 'Experience', 'Skills', and 'Interests'. Below the profile, there are tabs for 'More about me' and 'My events'. The 'My events' tab is active, showing four categories: 'I applied to:', 'I am accepted to:', 'I was rejected from:', and 'I saved:'. Each category currently shows 'No events'.

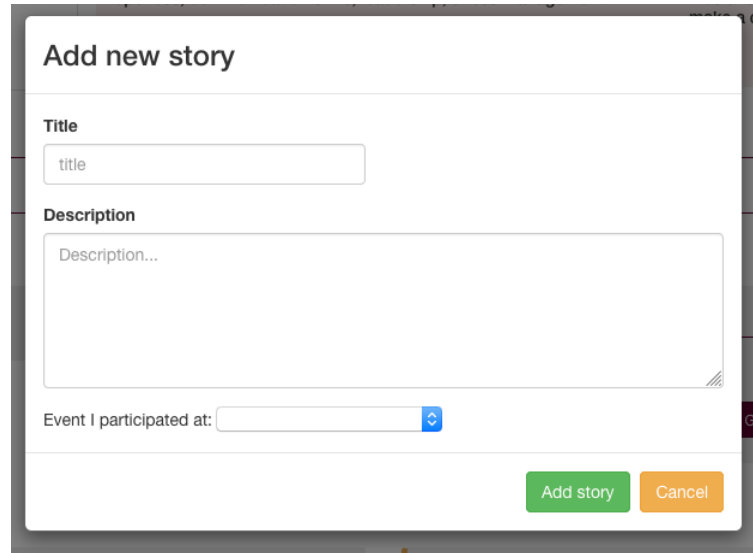
Figure 7.9 Volunteer-profile page

• Organization-Profile Page

After authentication as an organization user, the organization-profile page is displayed (figure 7.10). Same as the volunteers, organizations are able to modify the previously provided information. The organization user can add new events, edit the existing events or delete them. The modal used for editing an event is shown in the figure 7.12. The presented events can be filtered using the calendar component. Each event can be separately managed by the organization from the **Manage Event Page**, accessible by clicking the Manage button. Other authenticated users (volunteers and organizations) can visit the organization user profile.

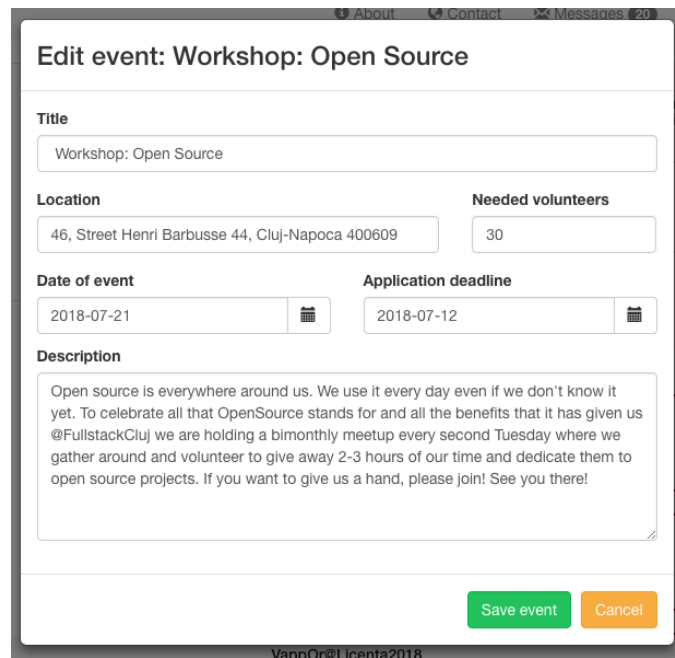
The screenshot shows an organization profile for 'CodingLife', established since June 16, 2017. The profile includes a logo, an 'Edit profile' link, and sections for 'Our story', 'Activity domains', 'Facebook', 'Address', 'Website', and 'Email'. Below the profile, there is a calendar for July 2018 and a section titled 'Our latest events'. The calendar shows the dates from July 1 to July 31. The 'Our latest events' section lists two events: 'Workshop: Open Source' (July 21, 2018) and 'Coding life' (July 1, 2018). Each event has buttons for 'Edit', 'Manage', and 'Delete', and statistics for 'Accepted', 'Rejected', 'Required', and 'Applied'.

Figure 7.10 Organization-profile page



The 'Add new story' modal form is a white rectangular box with a dark border. It has a title 'Add new story' at the top. Below the title, there is a 'Title' label followed by a text input field containing the placeholder 'title'. Underneath is a 'Description' label followed by a larger text area with the placeholder 'Description...'. At the bottom, there is a label 'Event I participated at:' followed by a dropdown menu with a blue arrow icon. At the very bottom right, there are two buttons: a green 'Add story' button and an orange 'Cancel' button.

Figure 7.11 Add story modal



The 'Edit event: Workshop: Open Source' modal form is a white rectangular box with a dark border. It has a title 'Edit event: Workshop: Open Source' at the top. Below the title, there is a 'Title' label followed by a text input field containing 'Workshop: Open Source'. Underneath, there are two columns: 'Location' with a text input field containing '46, Street Henri Barbusse 44, Cluj-Napoca 400609' and 'Needed volunteers' with a text input field containing '30'. Below these, there are two columns: 'Date of event' with a date input field containing '2018-07-21' and a calendar icon, and 'Application deadline' with a date input field containing '2018-07-12' and a calendar icon. At the bottom, there is a 'Description' label followed by a text area containing the text: 'Open source is everywhere around us. We use it every day even if we don't know it yet. To celebrate all that OpenSource stands for and all the benefits that it has given us @FullstackCluj we are holding a bimonthly meetup every second Tuesday where we gather around and volunteer to give away 2-3 hours of our time and dedicate them to open source projects. If you want to give us a hand, please join! See you there!'. At the very bottom right, there are two buttons: a green 'Save event' button and an orange 'Cancel' button.

Figure 7.12 Edit event modal

- **Manage Event Page**

The event management process can be splitted into four phases, as emphasized by the design of the Manage Event page:

1. **Check volunteers phase** - is the phase when the volunteers which applied to the event must be accepted or rejected, as it is shown in the figure 7.13; the volunteers' profiles can be accessed by just clicking their name.

Phase 1: Check volunteers

Applied (35)	Accepted (20)	Rejected (5)
Dariana Lupea ✓ ✗	Darius Popescu ✗	Ana-Maria Dobre ✓
Paula Muresan ✓ ✗	Bogdan Marius ✗	Lucian Gheorghe ✓
Alin-Andrei Stoica ✓ ✗	Alin-Andrei Stoica ✗	Alina Rusu ✓

Figure 7.13 Check volunteers phase

2. **Add new task phase** - for each task, the required data (title, location, description, selected volunteers, date, time interval) must be entered. Tasks can be assigned only to accepted (available) volunteers. The figure 7.14 presents the template for adding a task.

Phase 2: Add new task

Title

Location

Description

Available volunteers
 Darius Popescu
 Bogdan Marius
 Alin-Andrei Stoica
 Rares Cristian Silviu

Date of event

Hour interval

: AM
 : AM

Figure 7.14 Add task phase

3. **Manage tasks phase** – as shown in the figure 7.15, the created tasks can be visualized, edited, deleted and feedback can be sent to volunteers.

Phase 3: Manage tasks

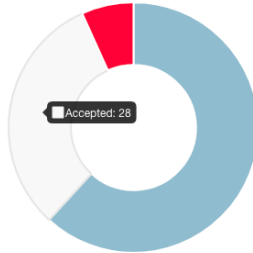
Date	Status	Title	Location	Start Hour	End Hour	Operation
16 July 2018	TODO	Offer guidance to participants	Str. Horea, nr. 97, Cluj-Napoca	9:00 AM	10:00 PM	
05 March 1974	TODO	Print flyers	Deisler House, Str. Avram Iancu	9:00 AM	9:00 AM	
16 July 2018	TODO	Prepare the meeting room	Str. Horea, nr. 97, Cluj-Napoca	8:20 PM	10:57 AM	

Figure 7.15 Manage tasks phase

4. **Generate charts phase** - charts are generated in order to offer a better overview of the event's dynamic and volunteers' engagement. The first chart in the figure 7.16 offers an overview of the number of volunteers who applied to event, who were accepted or rejected. The second chart shows the number of tasks which have a specific status.

Phase 4: Generate charts

Volunteers per event



Task status

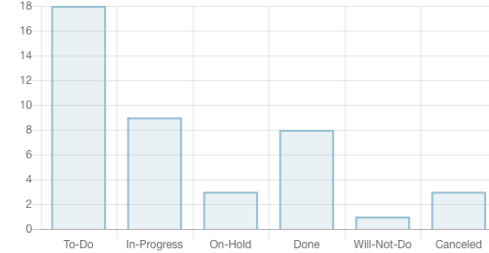


Figure 7.16 Generate charts phase

- **Settings page**

This page presents a functionality which allows the user to modify the account's password. The current password is required in order to successfully perform this operation, as presented in the figure 7.17.

Change password

Current password

New password

Confirm new password

Save changes

Figure 7.17 Change password on Settings page

Chapter 8. Conclusions

This chapter will present the conclusion of the paper, by offering an overview of the progress made in accomplishing the proposed objectives. Ideas for further development will be shared, since the obtained results could be improved and the application's functionalities completed.

8.1. Achievements

The main objective of the proposed project was to design and implement a web information system which supports the organizations in the volunteers selection and management processes and provides efficient communication mechanisms. The developed web application represents a powerful tool which can be used by **volunteers** who search for volunteering opportunities and **organizations** which promote their projects, recruit volunteers and manage them. Moreover, the users of the application can use the chat functionality to exchange messages and they will be notified with regards to important events. Therefore, we can affirm that the main objective was accomplished.

Next, we will present how the **specific objectives** were achieved through the **features and functions** provided by the implemented system. An important aspect is the fact that the web application can be accessed by non-authenticated users (named guests), who are allowed to access the unrestricted pages. In order to gain more access, the **registration** is required and it is differentiated based on the user's type: volunteer or organization. If the registration is successful, the users are able to **authenticate** using the unique username and the associated password.

Organization and volunteer users have a **personal profile** which they can manage by modifying previously provided information. The organizations can **add events** to which volunteers may apply, whereas the volunteers may **create stories** about the events they participated at. A way of sharing ideas about the events is represented by the **comments section**, where authenticated users can add comments. The admin user is responsible with managing this area and is able to even block user accounts if necessary.

Event management and evaluation represents an important feature offered by the system, since it reduces the organizations'effort for the volunteer's management. Through our web application, organizations are able to accept and reject volunteers to a certain event, create tasks and manage them, send feedback to volunteers and generate useful charts.

Moreover, the system provides **communication and interaction mechanism** such as email, online messaging, notifications, which efficientize the exchange of information.

As presented above, the specific objectives which were mapped to the functional requirements of the system are covered. The realization of **general objectives**, represented by non-functional requirements, will be evaluated next.

Usability - The user experience offered by our application was carefully thought. The user interface is friendly and intuitive, guiding the user to accomplish the initiated tasks, as shown in the images from the section 7.3.2. As an example, the volunteer user is able to apply to an event from three different pages of the applications. This facility is

provided in order to facilitate the participation to event process. Moreover, users continuously receive feedback from the system through success and error messages.

Security - Protecting user data is an important objective of the system. This is done by using secret tokens for authentication and information exchange, thus restricting the access to some pages of the application and securely transmitting information between client and server.

Performance - In order to monitor the performance of the backend server, we used JavaMelody⁵⁴, a tool which measures and calculates statistics on real operation of an application, depending on its usage. By consulting the obtained statistics, it can be observed that the targeted product performance is satisfied.

Portability- The portability of the developed application is ensured through the fact that it is a web application and it can be opened in various web browsers. Therefore, its behavior was thoroughly tested in the following web browsers: Google Chrome, Mozilla Firefox, Safari and Opera.

8.2. Further development

Due to the fact that the application was developed on a modular principle and each functional component is located in a separate project module, the subsequent development of the application should not require a huge effort on the developer's side.

The implemented volunteer recruitment and management system provides the basic features and functions required from a system of this type. Therefore, many possibilities of further development can be identified in order to improve the existing functions and to add more value to the application's business. We will present next some ideas which could be used as a starting point for enhancing the developed web application.

- **Membership option:** the volunteers could become members of one or several organizations and will be notified whenever a new event is added by that organizer. Moreover, the organizations could bring their own members on the platform and could easily differentiate them from other volunteers. This idea could be implemented by saving the connections between volunteers and associated organizations.
- **Volunteer availability:** allowing volunteers to define their availability would facilitate the task assignment process, while the possibility of directly accepting or declining a task would reduce the volunteer manager's work. The volunteer could just pick an hour interval from a list of options when applying to an event.
- **Management of large events:** the current implementation targeted relatively small events, which are organized during a single day. In order to cover events which last more than one day, an event should be added for each day. By adding a date interval or integrating the idea of **subevents**, our platform could support the management of larger events.
- **Groups of volunteers per event:** viewing who the other participants to events are, would increase the volunteers' interest in the event. If groups containing the volunteers who participate to the same event would exist, they could easily

⁵⁴ <https://github.com/javamelody/javamelody/wiki>

interact in order to clarify event related aspects, without asking the organizer's help. The chat functionality might be used for implementing this idea.

- **Participant user:** besides the volunteer and organization users, another type of user could be added - participant - such that the application could be used also by the participants to the events, who could visualize details and confirm their participation
- **Social media integration:** would simplify the authentication, if volunteer users could use their Gmail and Facebook accounts in order to register and login. Moreover, if the events could be shared on Facebook, Twitter or other social media environments, more users will be attracted on the platform. Such APIs exist and they could easily be integrated into the project.

An idea which is costly, from a perspective of time and human resources, but would bring a huge benefit to the entire project, is the implementation of a mobile application to be used mainly by volunteers. They could “clock-in” and “clock-out” when working on the assigned tasks, using the location provided through GPS. Therefore, their involvement would be quantifiable and the volunteer evaluation process could be improved. Moreover, they would be able to change the status of a task from “to-do” into “in-progress” or other, improving the task's progress visibility and offering organizations the possibility of keeping track of volunteer's activity.

Bibliography

[1] Anca Nastase, “Despre măsurarea impactului voluntariatului în România”, SlideShare, 2016

[Online]. Available: <https://www.slideshare.net/inscomunicare/despre-msurarea-impactului-voluntariatului-n-romnia>

[2] Wilson John, “Volunteering. Annual Review of Sociology”, August 2000, Vol. 26:215-240, DOI:10.1146/annurev.soc.26.1.215

[3] Jeffrey L. Brudney, Lucas C.P.M. Meijs, “Models of Volunteer Management: Professional Volunteer Program Management in Social Work”, Human Service Organizations Management, Leadership & Governance, 2014, 38:3, 297-309, DOI: 10.1080/23303131.2014.899281

[4] J. Schönböck, M. Raab, J. Altmann, E. Kapsammer, A. Kusel, B. Pröll, W. Retschitzegger, W. Schwinger, “A Survey on Volunteer Management Systems”, 2016 49th Hawaii International Conference on System Sciences, DOI 10.1109/HICSS.2016.100

[5] Hung-Yi Chen, Yueh-Chin Chen, Huei-Ling Li, Hsiao-Chun Wu, “Developing volunteer management system with Java EE Technology: The case of Taichung volunteer service promotion center”, 2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST), DOI: 10.1109/ICAWSST.2017.8256433

[6] Shuangli Wang, “A research on motivating and managing volunteers for large-scale sports events in China”, 2009 ISECS International Colloquium on Computing, Communication, Control, and Management, DOI: 10.1109/CCCM.2009.5267855

[7] Jeni Warburton, Melissa Moore, Melanie Oppenheimer, “Challenges to the Recruitment and Retention of Volunteers in Traditional Nonprofit Organizations: A Case Study of Australian Meals on Wheels”, International Journal of Public Administration 0:0, 2017, pages 1-13

[8] Harshad B. Prajapati, Vipul K. Dabhi, “High Quality Web-Application Development on Java EE Platform”, Advance Computing Conference, 2009. IACC 2009. IEEE International, DOI: 10.1109/IADCC.2009.4809267

[9] Y. Furukawa, “Web-based control application using WebSocket”, in Proceedings of ICALEPCS 2011, Grenoble, France

[10] Kyle Andrei, Chris Bernard, Jay Leslie and Laura Quinn, “A Consumers Guide to Software for Volunteer Management”, May 2011, pp. 4-12.
[Online]. Available:

<https://www.techsoup.org/support/articles-and-how-tos/consumers-guide-to-software-volunteer-management>

[11] Steven Hall, Chris Milway, Helen Ridgway, Jack Garfinkel, Tony Goodrow, Myles Kunzli, “How to choose a volunteer management system”, Knowhow Nonprofit, November 2017

[Online]. Available:

<https://knowhownonprofit.org/how-to/how-to-choose-a-volunteer-management-system>

[12] Gabriel L. Muller, “HTML5 WebSocket protocol and its application to distributed computing”, Master’s thesis, Cranfield University, School of engineering, 2014

[13] Deepak Kumar, “Best Practices for Building RESTful Web services”, Infosys Limited, Bengaluru, India, 2017.

[Online]. Available:

<https://www.infosys.com/digital/insights/Documents/restful-web-services.pdf>

[14] Thomas Connolly, Carolyn Begg, “Database systems - A practical Approach to Design, Implementation, and Management”, Sixth Edition, Pearson, 2015

[15] George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, “Distributed systems concepts and design”, Fifth Edition, Addison Wesley, 2012

[16] Karl Wieggers, Joy Beatty, “Software Requirements”, 3rd Edition, Microsoft Press, 2013

[17] Kathy Sierra, Bert Bates, “Head First Java”, Second Edition, O’Reilly Media, 2005

[18] Ovidiu Pop, Information Systems, Lecture Notes, TUCN, Faculty of Automation and Computer Science, 2018

[19] Best Volunteer Management Software in 2018, Capterra Reviews

[Online]. Available: <https://www.capterra.com/volunteer-management-software/>

[20] Samaritan technologies - Volunteer software

[Online]. Available: <https://samaritan.com/>

[21] Spring Framework Online Documentation

[Online]. Available: <https://docs.spring.io/spring/docs/current/spring-framework-reference>

[22] AngularJS Online Documentation

[Online]. Available: <https://docs.angularjs.org/guide/concepts>

[23] JSON Web Token Online Documentation

[Online]. Available: <https://jwt.io>

Appendix 1: List of figures

Figure 3.1 Profile and Task packages.....	12
Figure 3.2 Evolution Package.....	12
Figure 4.1 Guest Use Case UML Diagram	21
Figure 4.2 Volunteer Use Case UML Diagram	22
Figure 4.3 Organization Use Case UML Diagram	23
Figure 4.4 Administrator Use Case UML Diagram	24
Figure 4.5 Create task use case flow diagram	27
Figure 4.6 System architecture	28
Figure 4.7 Spring Framework overview.....	30
Figure 4.8 RESTful communication in Spring MVC.....	31
Figure 4.9 Maven project structure	32
Figure 4.10 Maven's default lifecycle phases	32
Figure 4.11 JWT security flow	34
Figure 4.12 Example of expression in AngularJS	37
Figure 4.13 Data binding in AngularJS templates.....	37
Figure 4.14 WebSockets communication flow	39
Figure 4.15 Geocoding API request format	40
Figure 4.16 Google Maps with pin-point on Cluj-Napoca.....	40
Figure 5.1 Detailed architecture of front-end component	43
Figure 5.2 AngularUI routing.....	43
Figure 5.3 Guest's navigation diagram	44
Figure 5.4 Authenticated user's navigation diagram.....	44
Figure 5.5 Event Controller definition	44
Figure 5.6 EventService definition.....	45
Figure 5.7 UserService definition.....	45
Figure 5.8 Layered architecture pattern.....	46
Figure 5.9 Backend package diagram	48
Figure 5.10 Definition of TaskController.....	49
Figure 5.11 Definition of EventService.....	50
Figure 5.0.12 Implementation of Repository interface for VolunteerProfile entity	51
Figure 5.13 Classes interaction.....	52
Figure 5.14 JWT check of user's role	52
Figure 5.15 Cors Filter implementation	53
Figure 5.16 User object mapped with DozerBeanMapper	53
Figure 5.17 Database diagram	54
Figure 6.1 Test automation pyramid	60
Figure 6.2. Example of API call using Postman	61
Figure 7.1 Bitbucket repositories overview.....	62
Figure 7.2 Database configuration in application.properties.....	63
Figure 7.3 Sitemap.....	64
Figure 7.4 Homepage	65
Figure 7.5 Login modal	66
Figure 7.6 Volunteer register modal.....	66
Figure 7.7 Organization register modal	66

Figure 7.8 Detailed event page	67
Figure 7.9 Volunteer-profile page	68
Figure 7.10 Organization-profile page	68
Figure 7.11 Add story modal	69
Figure 7.12 Edit event modal	69
Figure 7.13 Check volunteers phase.....	70
Figure 7.14 Add task phase	70
Figure 7.15 Manage tasks phase.....	70
Figure 7.16 Generate charts phase.....	71
Figure 7.17 Change password on Settings page.....	71

Appendix 2: List of tables

Table 3.1 Comparative analysis of the similar systems	16
Table 4.1 Functional requirements	18
Table 4.2 Actors and responsibilities description.....	20
Table 6.1 Steps of test case 1.....	58
Table 6.2 Steps of test case 2.....	59

Appendix 3: Glossary

Term	Definition
Organization	In the context of this system, it denotes a user type and is represented by an organization member. Through organization we refer to an ONG, a society, a community or a self- organized group of persons organizing an event.
Event	It represents an activity proposed by an organization where volunteers can apply in order to achieve a common goal. The event details (title, description, date, application deadline, number of volunteers required, etc.) are specified by the organization.
HTTP	H yper T ext T ransfer P rotocol is an application protocol used for specifying the rules for data communication in World Wide Web.
REST	R Epresentational S tate T ransfer is an architectural style for providing communication standards between computer systems on the web.
API	A pplication P rogramming I nterface represents a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.
WebSockets	Is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server.
JSON	J avascript O bject N otation is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types.
STOMP	S imple T ext O riented M essage P rotocol, formerly known as TTMP, is a simple text-based protocol, designed for working with message-oriented middleware (MOM).

Appendix 4: UML Class diagram

