Content

[1 Abstract 6](#_Toc430422348)

[2 Goal and tasks 6](#_Toc430422349)

[3 Design and description of the proposed solution 6](#_Toc430422350)

[3.1 Software system requirements 7](#_Toc430422351)

[3.2 Logical model 7](#_Toc430422352)

[3.2.1 Use case diagram 8](#_Toc430422353)

[3.2.2 Activity diagram 11](#_Toc430422354)

[3.2.3 Package diagram 22](#_Toc430422355)

[3.2.4 Class diagram 23](#_Toc430422356)

[3.2.5 Sequence diagram 27](#_Toc430422357)

[3.3 System architecture 30](#_Toc430422358)

[3.4 Data organization – conceptual schema 30](#_Toc430422359)

[3.5 Implementation 32](#_Toc430422360)

[3.5.1 Data structure 32](#_Toc430422361)

[3.5.2 Technologies 34](#_Toc430422362)

[3.5.3 Description of main modules 37](#_Toc430422363)

[3.5.4 Structure and organization of user interface 42](#_Toc430422364)

[3.5.5 Format of input and output documents 43](#_Toc430422365)

[4 Instructions for use 51](#_Toc430422366)

[4.1 User guide 51](#_Toc430422367)

[4.2 Instructions and requirements for installation of the system 61](#_Toc430422368)

[4.3 Instructions for maintenance of the system 62](#_Toc430422369)

[4.4 Hardware requirements 62](#_Toc430422370)

[5 Test Results 62](#_Toc430422371)

[6 Main results 63](#_Toc430422372)

[7 Conclusions and recommendations 63](#_Toc430422373)

[8 References 64](#_Toc430422374)

[9 Attachments 65](#_Toc430422375)

figures

[Figure 1 Class diagram of UML [2] 7](#_Toc429944829)

[Figure 2 Use case diagram of mission management system of "NoBug's Snack Bar" 8](#_Toc429944830)

[Figure 3 Use case "Manage "Explanations” " 9](#_Toc429944831)

[Figure 4 Use case "Commands" 9](#_Toc429944832)

[Figure 5 Use case "Manage “Customers” " 9](#_Toc429944833)

[Figure 6 Use case "Manage “Orders” " 10](#_Toc429944834)

[Figure 7 Use case "Manage "Hints – build condition" " 10](#_Toc429944835)

[Figure 8 Use case "Manage "Settings" " 11](#_Toc429944836)

[Figure 9 Activity diagram "Saving a mission" 12](#_Toc429944837)

[Figure 10 Activity diagram "Verifying the input data" 13](#_Toc429944838)

[Figure 11 Marshalling 14](#_Toc429944839)

[Figure 12 Image to imghex conversion 16](#_Toc429944840)

[Figure 13 Build a condition 18](#_Toc429944841)

[Figure 14 Sub-activity "Build sub-condition" 19](#_Toc429944842)

[Figure 15 Activity diagram "Add/Edit explanation page" 20](#_Toc429944843)

[Figure 16 Activity diagram "Edit customer" 21](#_Toc429944844)

[Figure 17 Package diagram of the system 22](#_Toc429944845)

[Figure 18 Class diagram of package "dao" 23](#_Toc429944846)

[Figure 19 Class diagram of package "jdbc" 24](#_Toc429944847)

[Figure 20 Class diagram of package "missionmanager" 24](#_Toc429944848)

[Figure 21 Class diagram of package "mission" 24](#_Toc429944849)

[Figure 22 Class diagram of package "model" 25](#_Toc429944850)

[Figure 23 Class diagram of package "servlets" 25](#_Toc429944851)

[Figure 24 Class diagram of package "util" 25](#_Toc429944852)

[Figure 25 Class diagram of package "control" 26](#_Toc429944853)

[Figure 26 Class diagram of package "converter" 26](#_Toc429944854)

[Figure 27 Class diagram of package "listeners" 26](#_Toc429944855)

[Figure 28 Class diagram of package "i18n" 26](#_Toc429944856)

[Figure 29 Sequence diagram "Save mission" 27](#_Toc429944857)

[Figure 30 Sequence diagram "Mission verification" 28](#_Toc429944858)

[Figure 31 Sequence diagram "Image to "text" conversion" 29](#_Toc429944859)

[Figure 32 System architecture 30](#_Toc429944860)

[Figure 33 Conceptual schema 31](#_Toc429944861)

[Figure 34 Database schema 32](#_Toc429944862)

[Figure 35 Top 10 Web frameworks [16] 36](#_Toc429944863)

[Figure 36 JSF Component libraries - popularity comparison 37](#_Toc429944864)

[Figure 37 Structure of UI 42](#_Toc429944865)

[Figure 38 Sitemap 43](#_Toc429944866)

[Figure 39 User interface 51](#_Toc429944867)

[Figure 40 Add new page 52](#_Toc429944868)

[Figure 41 Explanation page - filled out with no errors 53](#_Toc429944869)

[Figure 42 Explanation page - invalid image path or format 53](#_Toc429944870)

[Figure 43 Editing/Removing pages 54](#_Toc429944871)

[Figure 44 Commands 54](#_Toc429944872)

[Figure 45 Customers in "Snack Bar" 55](#_Toc429944873)

[Figure 46 Customer dialog 56](#_Toc429944874)

[Figure 47 Condition dialog 57](#_Toc429944875)

[Figure 48 Function provider 57](#_Toc429944876)

[Figure 49 Condition list 58](#_Toc429944877)

[Figure 50 Settings 59](#_Toc429944878)

[Figure 51 Building block expressions 60](#_Toc429944879)

[Figure 52 Created mission "Test Misison" 61](#_Toc429944880)

[Figure 53 Issues - GitHub.com 62](#_Toc429944881)

tables

[Table 1 Table "Mission" 33](#_Toc429944821)

[Table 2 Table "ClassesMissions" 33](#_Toc429944822)

[Table 3 Table "FunctionProvider" 33](#_Toc429944823)

[Table 4 Table "FunctionProviderValues" 33](#_Toc429944824)

[Table 5 Table "Hints" 34](#_Toc429944825)

[Table 6 Table "HintCategories" 34](#_Toc429944826)

[Table 7 Table "Commands" 34](#_Toc429944827)

abbreviations

|  |  |
| --- | --- |
| **AI** | Auto Incremental |
| **API** | Application Programming Interfaces |
| **CD** | Compaq Disk |
| **CPU** | Central Processing Unit |
| **DB** | Database |
| **DBMS** | Database Management System |
| **DTD** | Document Type Definition |
| **DV** | Default Value |
| **FK** | Foreign Key |
| **GB** | Gigabyte |
| **HDD** | Hard Disk Drive |
| **IBM** | International Business Machines Corporation |
| **ID** | Identification Number |
| **IDE** | Integrated Development Environment |
| **JDK** | Java Development Kit |
| **JSF** | Java Server Faces |
| **NN** | Not Null |
| **OMG** | Object Management Group |
| **PK** | Primary Key |
| **RAM** | Random-Access Memory |
| **SCM** | Source Code Management |
| **SOX** | Simple Object XML |
| **UI** | User Interface |
| **UML** | Unified Modeling Language |
| **URL** | Uniform Resource Locator |
| **XML** | Extensible Markup Language |
| **XSD** | XML Schema Definition |
| **XSLT** | XML Transformations |

# Abstract

In this graduation project is discussed the developing process of a mission management system of “NoBug’s Snack Bar” (Part 1). Actually, the game is being developed as a game engine. This means that the missions and challenges are not predefined in the game, but are created by the teachers.

The game “NoBug’s Snack Bar” is designed for using by students to improve their skills for problem solving in introductory computer programming while the mission management system is designed for using by teachers.

The technologies involved in this work were Java Server Faces, PrimeFaces, Ajax, XML, HTML5 and MySQL.

***Keywords:*** *Java, PrimeFaces, JSF, XML, XSD, MySQL, Mission Management System, Expression Language, UML,* NoBug’s Snack Bar.

В този дипломен проект е разгледан процесът на разработване на система за управление на мисии в играта „NoBug’s Snack Bar“ (Част 1). Всъщност, играта се разработва като „game engine“. Това означава, че мисиите и предизвикателствата в нея не са предефинирани, а се създават.

Играта „NoBug’s Snack Bar“ е предназначена за използване в учебните занятия от учениците с цел подобряване уменията за решаване на проблеми в уводното компютърно програмиране, докато системата за управление на мисии е предназначена за използване от учители.

В този проект са използвани следните технологии: Java Server Faces, PrimeFaces, Ajax, XML, HTML5 and MySQL.

***Ключови думи:*** *Java, PrimeFaces, JSF, XML, XSD, MySQL, Mission Management System, Expression Language, UML,* NoBug’s Snack Bar.

# Goal and tasks

The main goal is to deploy a tool which users can easily use to create their own missions and maintain the available ones in the game – “NoBug’s Snack Bar”. For achieving it, it is necessary to perform the following tasks:

* To make interviews with the future users of the application (the so called target group) to clarify what their desires and expectations to it are;
* To define functional requirements of the application;
* To propose architectural and logical model of the application;
* To choose the most appropriated technologies for its implementation

# Design and description of the proposed solution

## Software system requirements

The software has to cover the following requirements:

* easy to understand and convenient to use user interface (UI);
* possibility for further expansions and additions;
* reliability and security;

## Logical model

The logical model is represented by Unified Modeling Language (UML). UML is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

UML (version 2) has many types of diagrams which are divided into two categories. Some types represent structural information, and the rest represent general types of behavior, including a few that represent different aspects of interactions. These diagrams can be categorized hierarchically as shown in the following class diagram (figure 1) [1]:

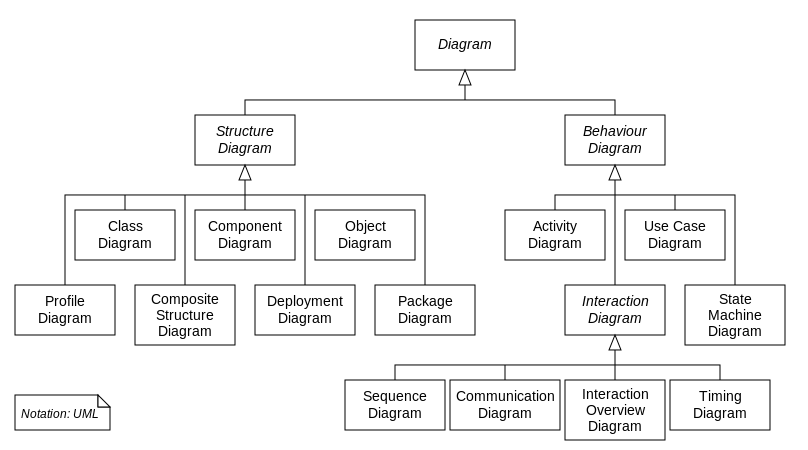


Figure 1 Class diagram of UML [2]

Diagrams used in this project are:

* Structural UML diagrams – class and package diagram;
* Behavioral UML diagrams – activity, sequence and use case diagram.

They are created using tools such as: Microsoft Visio Drawing 2013, Visual Paradigm 12.1 and ObjectAid UML explorer for Eclipse IDE (integrated development environment).

### Use case diagram

A use case diagram is a kind of UML diagram defined by Object Management Group (OMG), created for use case analysis. Use case diagram provides a graphical overview of goals (represented by use cases) users (represented by actors) want to achieve by using the system (represented by system boundary but is often opt out in diagram). Use cases in a use case diagram can be organized and arranged according to their relevance, level of abstraction and impacts to users. They can be connected to show their dependency, inclusion and extension relationships. The main purpose of modeling use case with use case diagram is to establish a solid foundation of the system by identifying what the users want [3].

The use case diagram of mission management system is shown in figure 2.

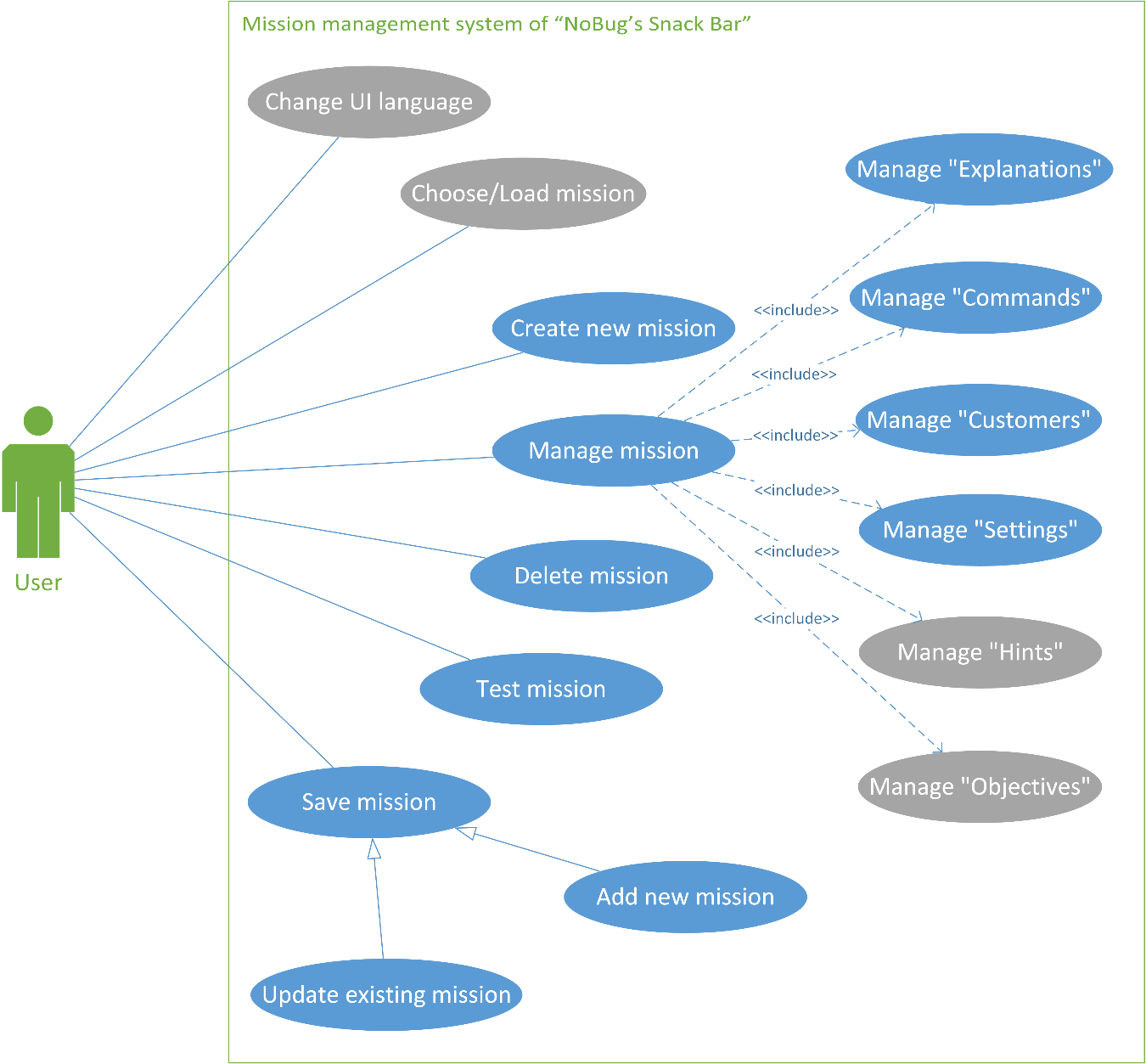


Figure 2 Use case diagram of mission management system of "NoBug's Snack Bar"

Because of the reason that the diagram has become too large and unreadable, some use cases are described in more detail in separate figures.

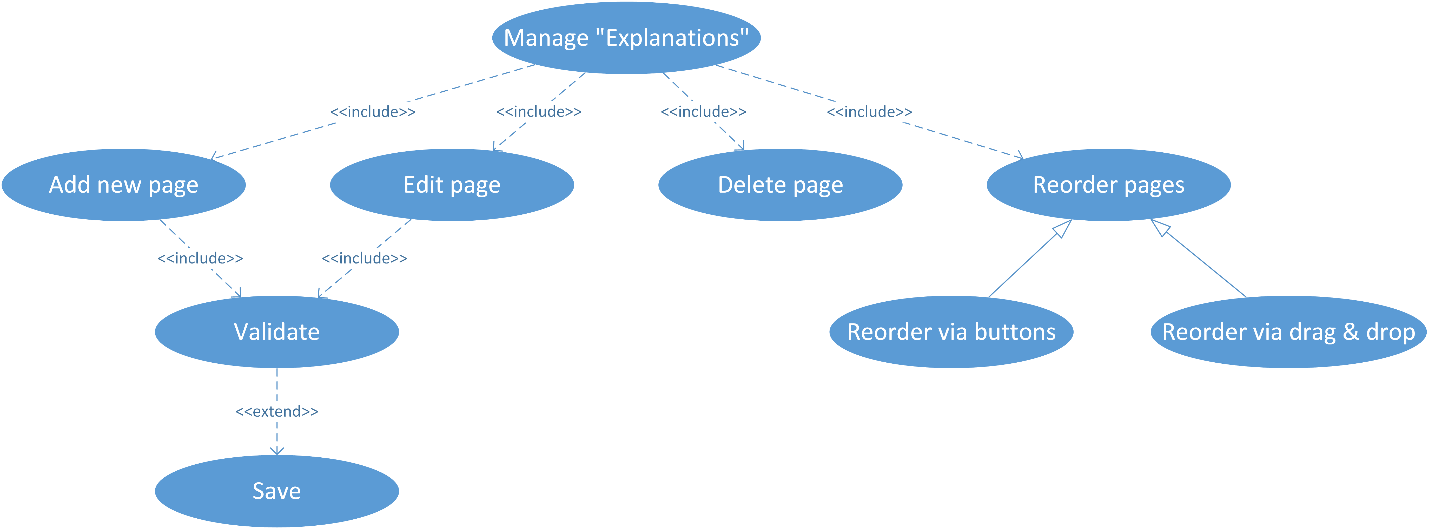


Figure 3 Use case "Manage "Explanations” "

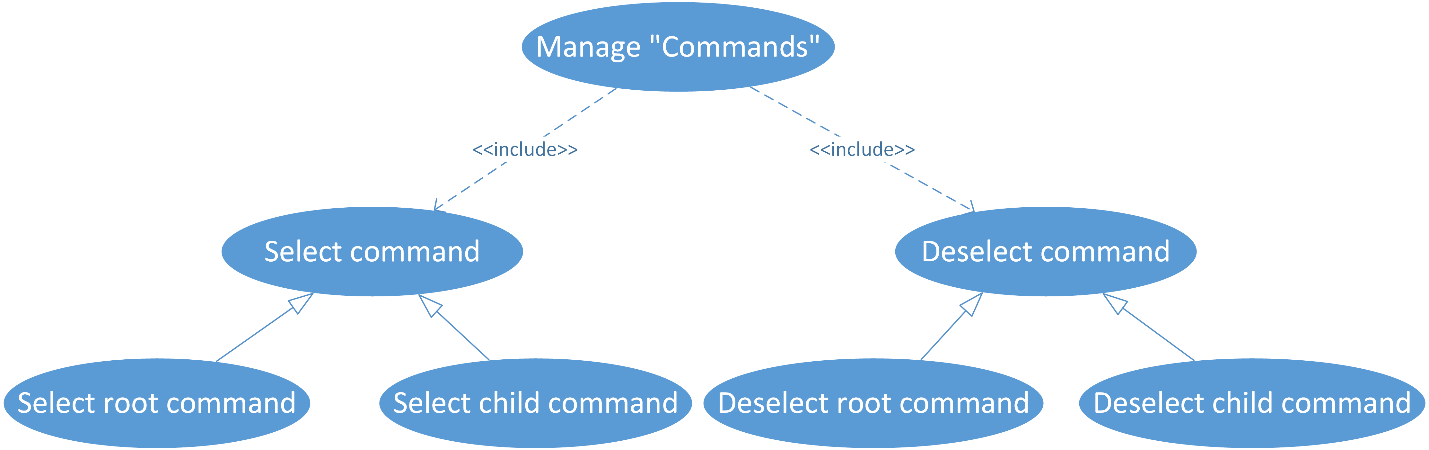


Figure 4 Use case "Commands"

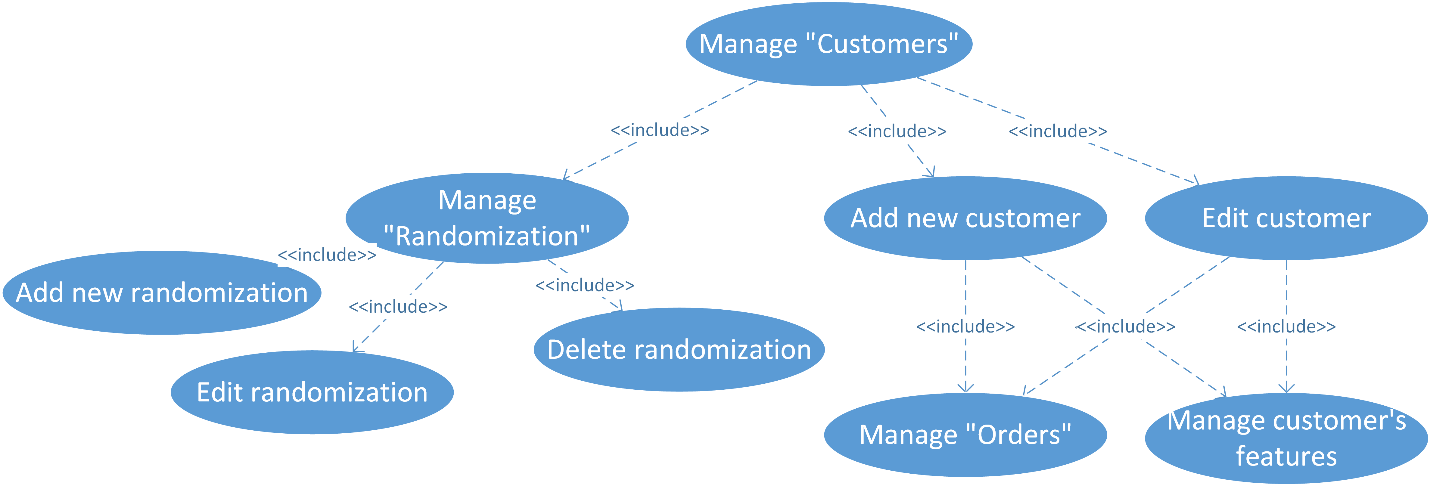


Figure 5 Use case "Manage “Customers” "

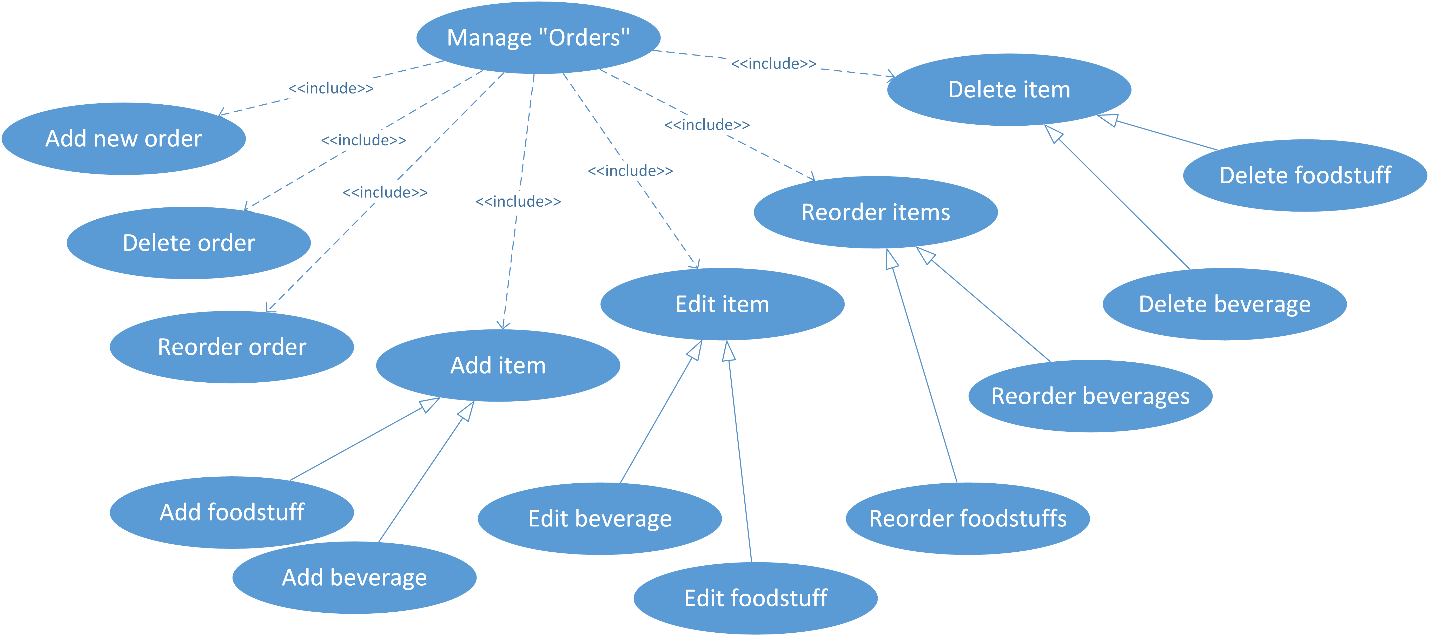


Figure 6 Use case "Manage “Orders” "

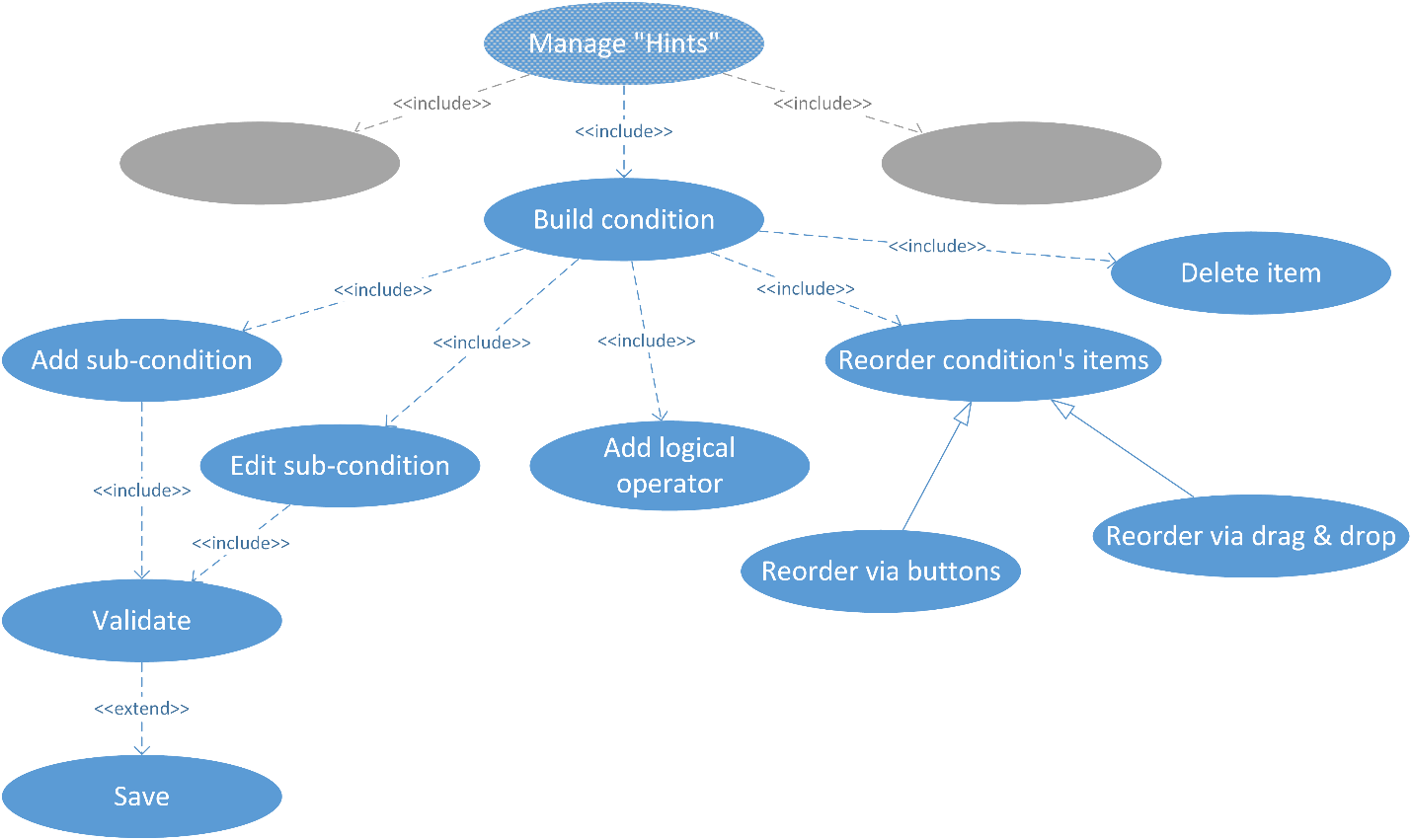


Figure 7 Use case "Manage "Hints – build condition" "

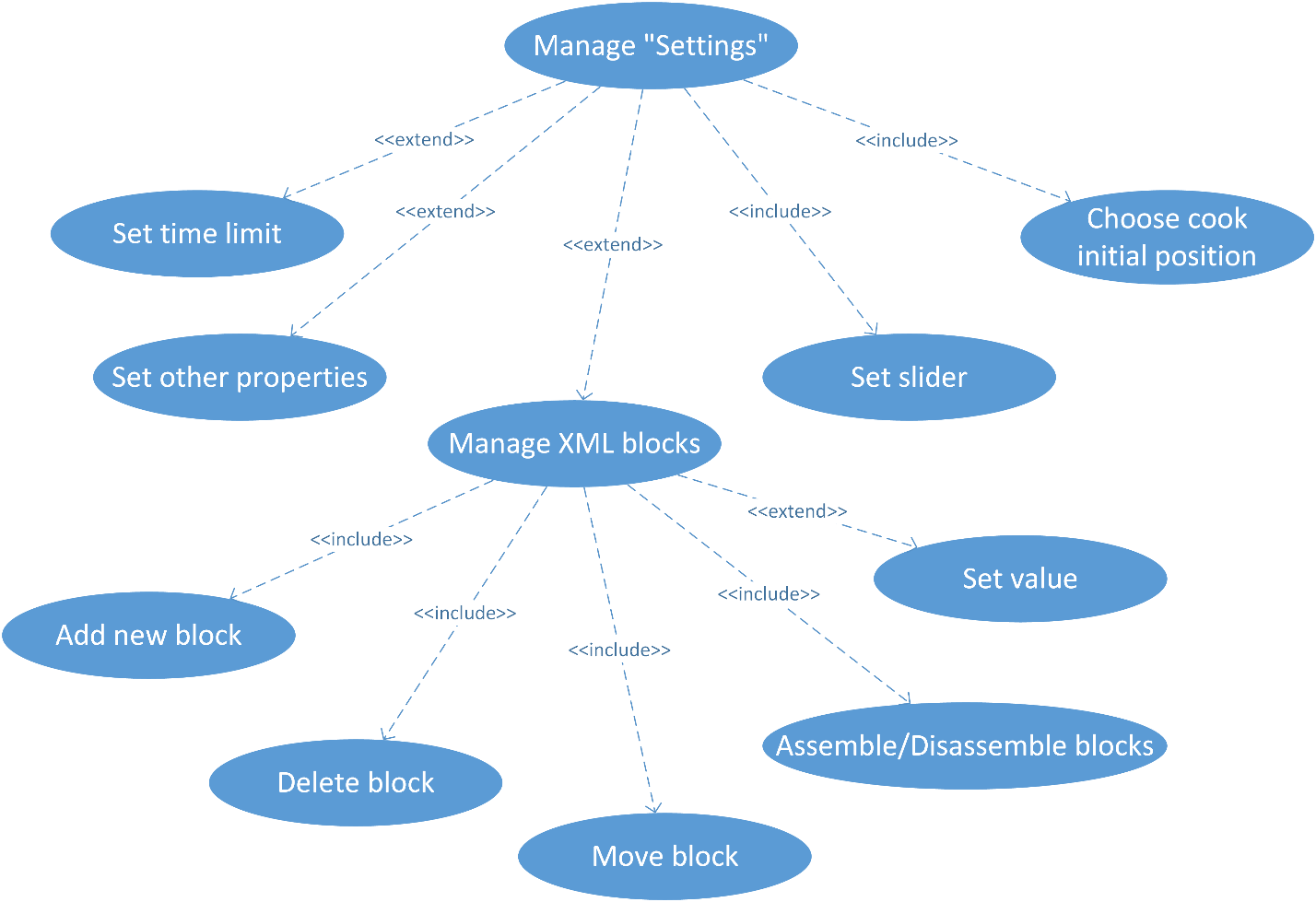


Figure 8 Use case "Manage "Settings" "

### Activity diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the UML, activity diagrams are intended to model both computational and organizational processes (i.e. workflows) [4].

An activity diagram is represented by shapes that are connected by arrows. Arrows run from activity start to completion and represent the sequential order of performed activities. Black circles represent an initial workflow state. A circled black circle indicates an end state. Rounded rectangles represent performed actions, which are described by text inside each rectangle.

A diamond shape is used to represent a decision, which is a key activity diagram concept. Upon activity completion, a transition (or set of sequential activities) must be selected from a set of alternative transitions for all use cases.

Synchronization bars indicating the start or completion of concurrent activities are used to represent parallel subflows [5].

In figure 9 is shown the activity diagram of saving a mission.

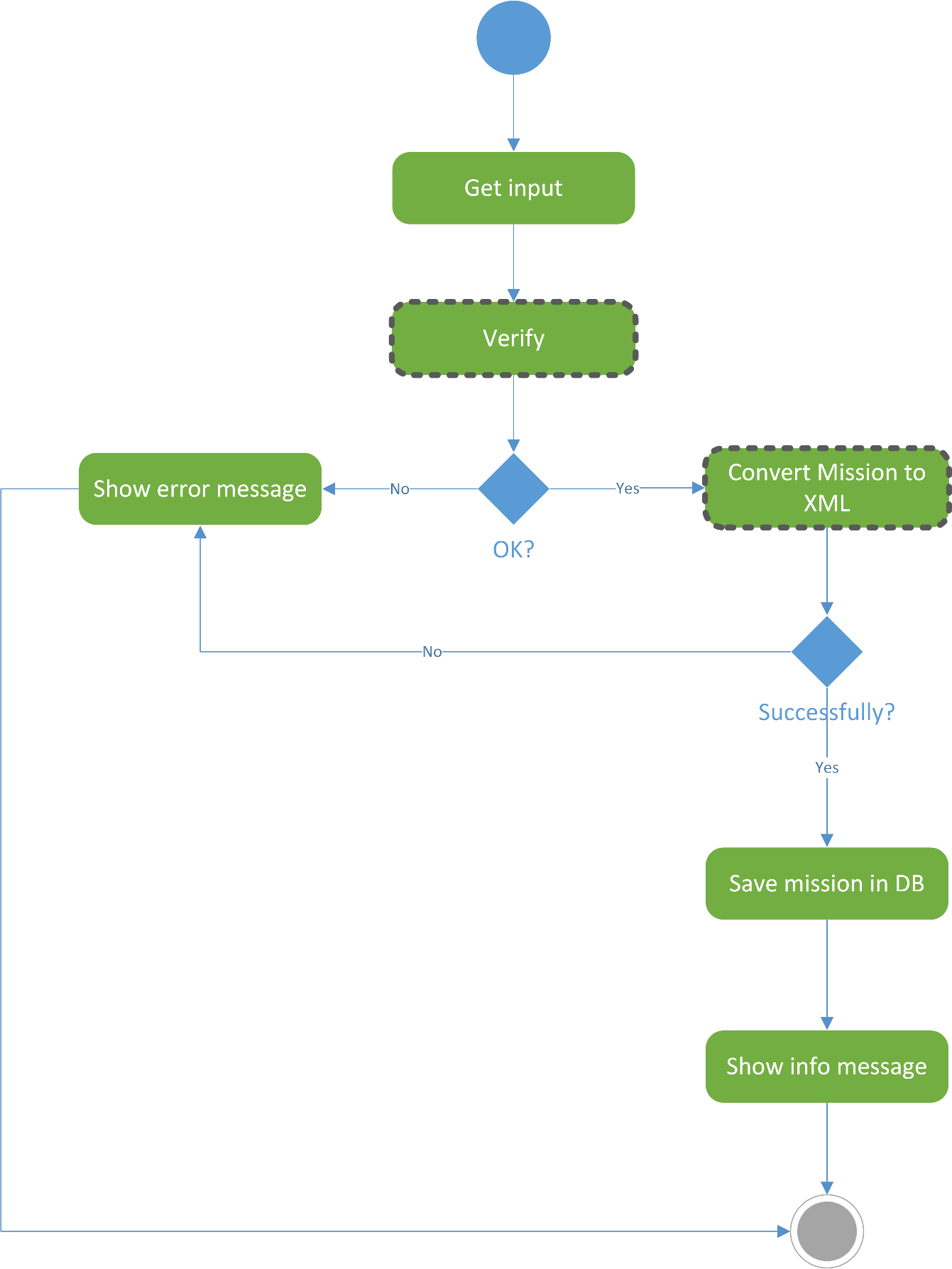


Figure 9 Activity diagram "Saving a mission"

First of all, user has to enter some input information, then the system checks (figure 10) if it is entered all required information (correctly). If the information is validated then the one will be converted (figure 11) to Extensible Markup Language (XML) and if this step is performed successfully, the data can be saved in database (DB). In case of an error in any of these steps the user will be notified by message, as well when the mission is saved successfully.

In figure 10 are shown the steps for verifying basic information, which is necessary for a mission to be converted and saved in DB:

* mission name;
* at least one customer;
* if the mission is not repeatable - at least one objective;
* at least one explanation page;
* at least one selected command or selected option “load blocks”.

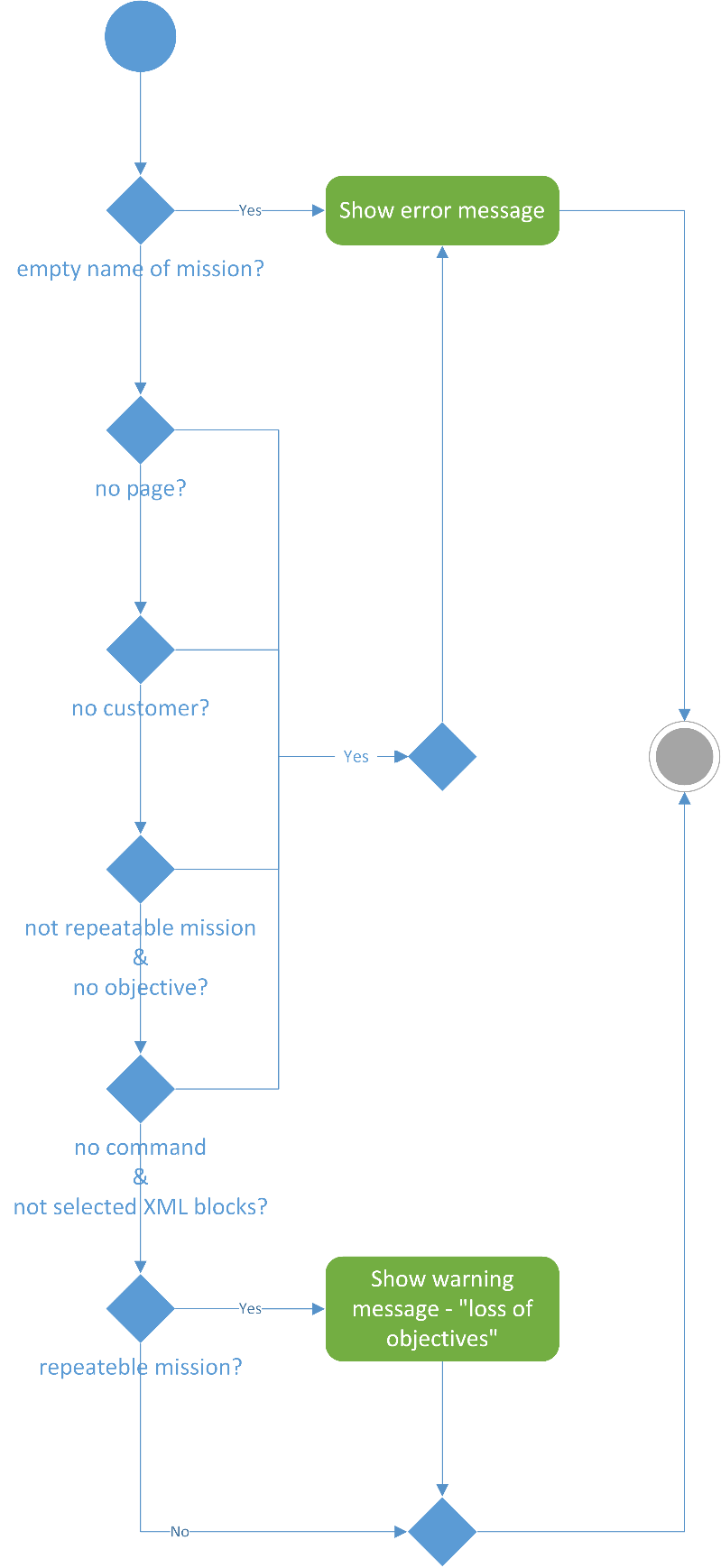


Figure 10 Activity diagram "Verifying the input data"

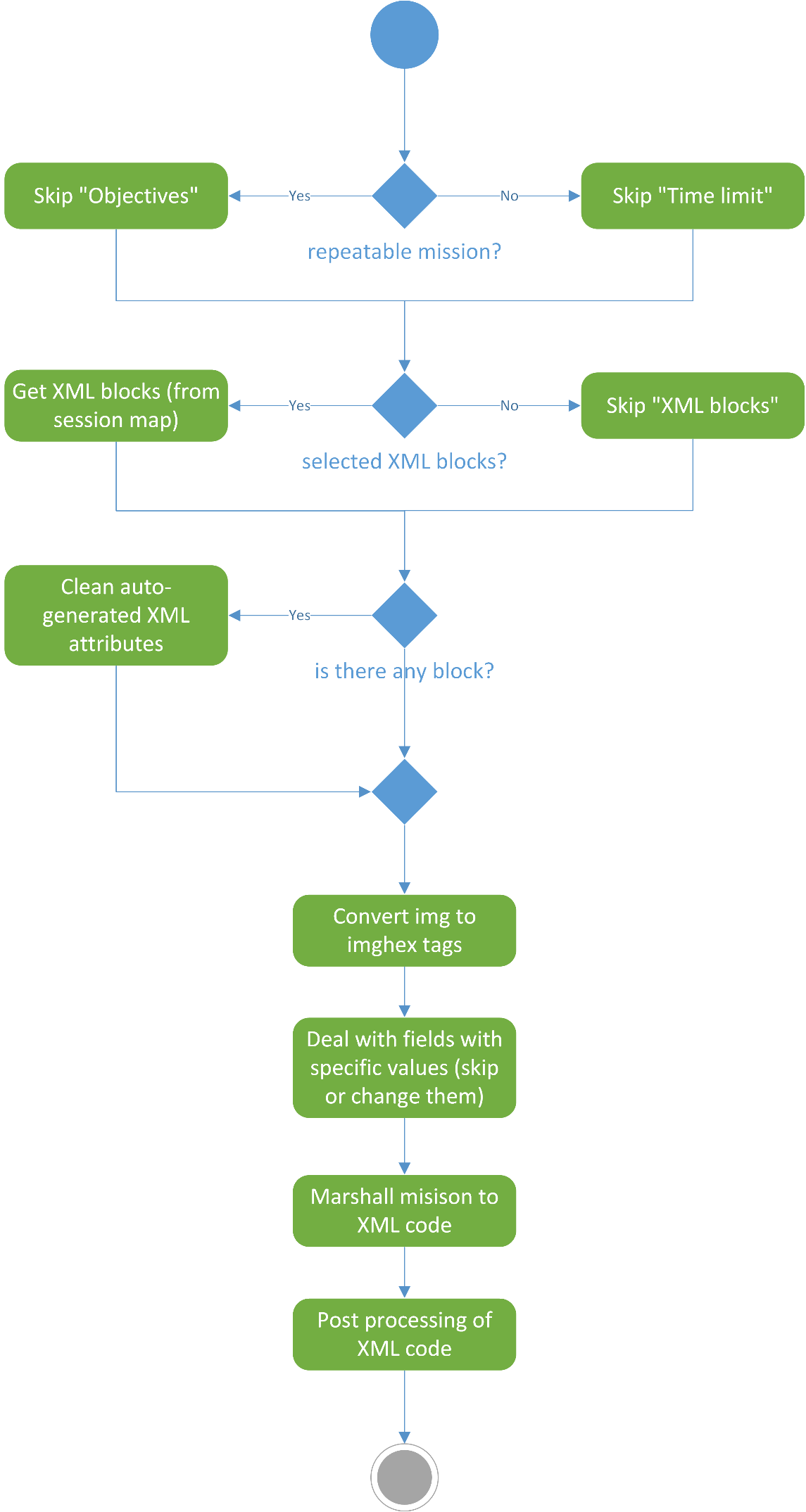


Figure 11 Marshalling

There are also some additional steps before a mission object to be converted to XML code (this action is called “marshalling”). These steps are not related with the “marshalling”, but with functional requirements of the app. For example, the XML code does not have to contain information for “objectives” if a mission is repeatable. On the other hand, if a mission is repeatable it does not have to contain “time limit” and so on.

If a user has not selected option “load blocks” then it can be skipped. Otherwise, it will be added. But it also has to be modified (add or remove some features), because it is auto-generated XML code by other tool.

Images in “explanation pages” (*and “hints”*) have to be represented in other form, because the XML code can contain only text. The activity diagram from figure 12 shows how to implement it.

After “marshalling”, it can be necessary, the XML code, to be processed. Some characters are reserved, such as less than (<) or greater than (>) signs. “Condition builder” feature uses these characters (figure 13), because of that they have to be replaced by entities. Character entities are used to display reserved characters in HTML. A character entity looks like this: *&entity\_name;* or *&#entity\_number*; [6].

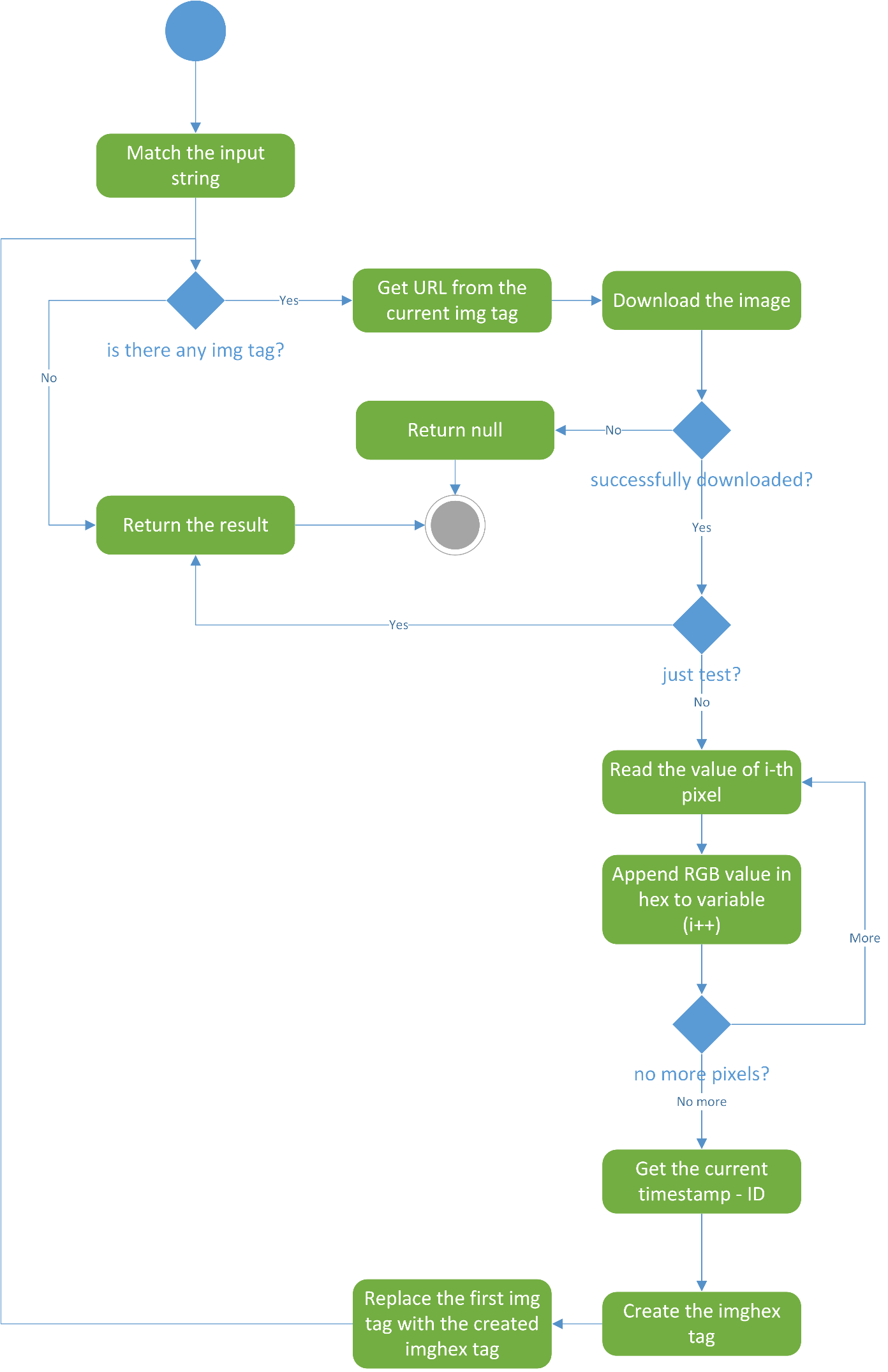
**

Figure 12 Image to imghex conversion

Conversion of image to sequence of hexadecimal numbers is illustrated in figure 12. At the beginning it is necessary to check if the input string contains image tag and get its URL (Uniform Resource Locator) address. Then, download the image file to local storage. If the file is successfully downloaded and its format is valid, it can be converted. Therefore, the image tag can be replaced by the new tag (imghex). But this activity can be used also for testing images whether they are valid. Because the conversion is necessary only when you save a mission.

For the purposes of this project is needed to create a tool, which allows building of conditional expressions. An conditional expression might contain logical operators, comparators and values for comparing. They have to be in strongly specific order: “<value1> comparator <value2> [*logical operator <value3> comparator <value4>…* ]”. Figures 13 and 14 show managing of all these things.

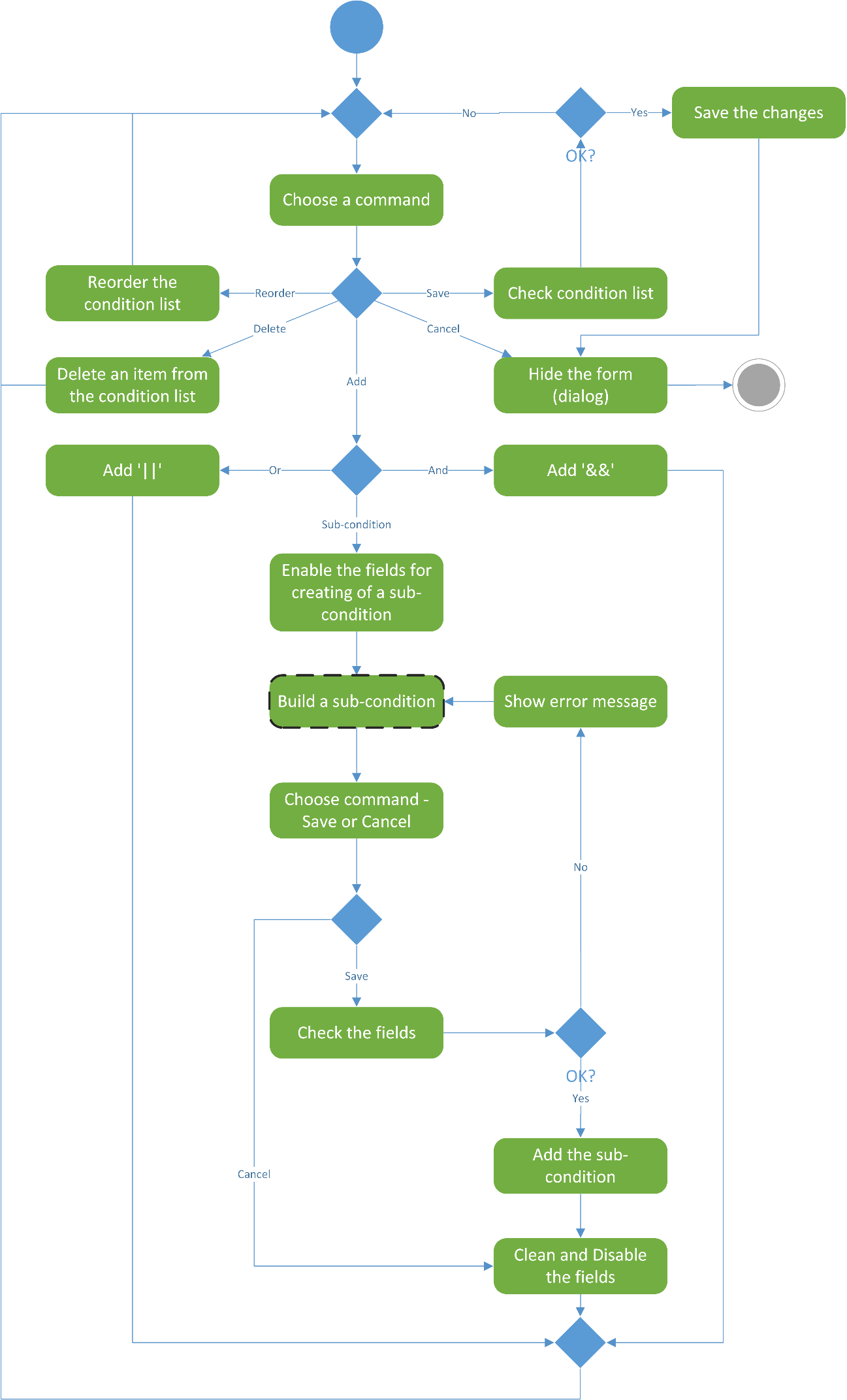


Figure 13 Build a condition

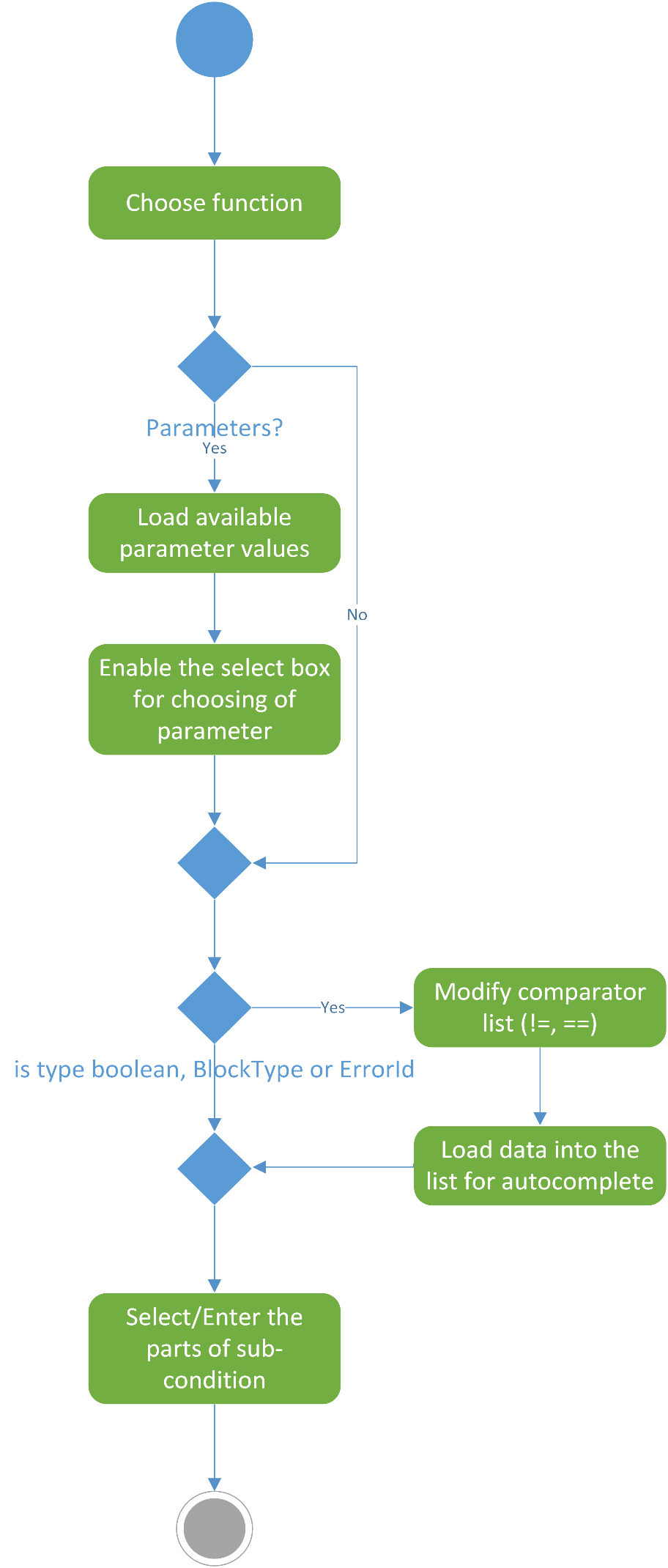


Figure 14 Sub-activity "Build sub-condition"

In figure 15 is shown the activity diagram for adding and editing of explanation pages. After choosing the command “Add page” or “Edit page” the form is available for the user. First of all, he has to enter some text then, save or cancel the operation. If he decides to save the changes it is necessary to validate the text information:

1. check if it is entered text;
2. if there are images they have to be checked also (figure 12).

If all checks went well, according to the selected command, the “page” will be added or updated.

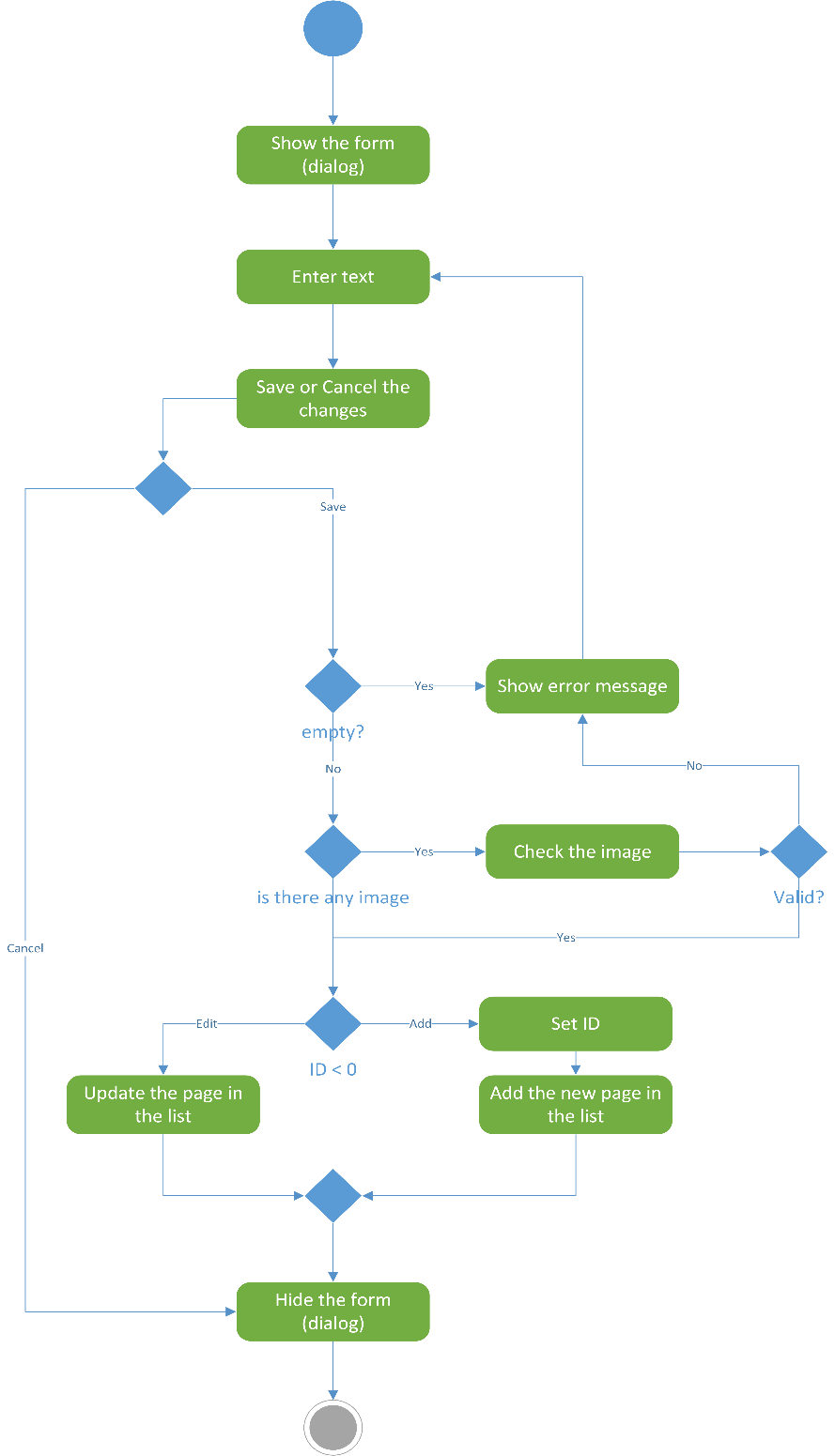


Figure 15 Activity diagram "Add/Edit explanation page"

The figure 16 shows how a customer can be added or his information be edited. The maximal number of customers is fixed – 12, so all the customers can be created in system initialization. User just has to select tables and chairs (place) for the customers. When he has selected a place, its identification number (ID) will be used for identifying which customer’s information to be shown. The decision, if a customer has to be involved in the mission, makes according to the number of orders of each customer. If a customer has at least one order he is “added”.

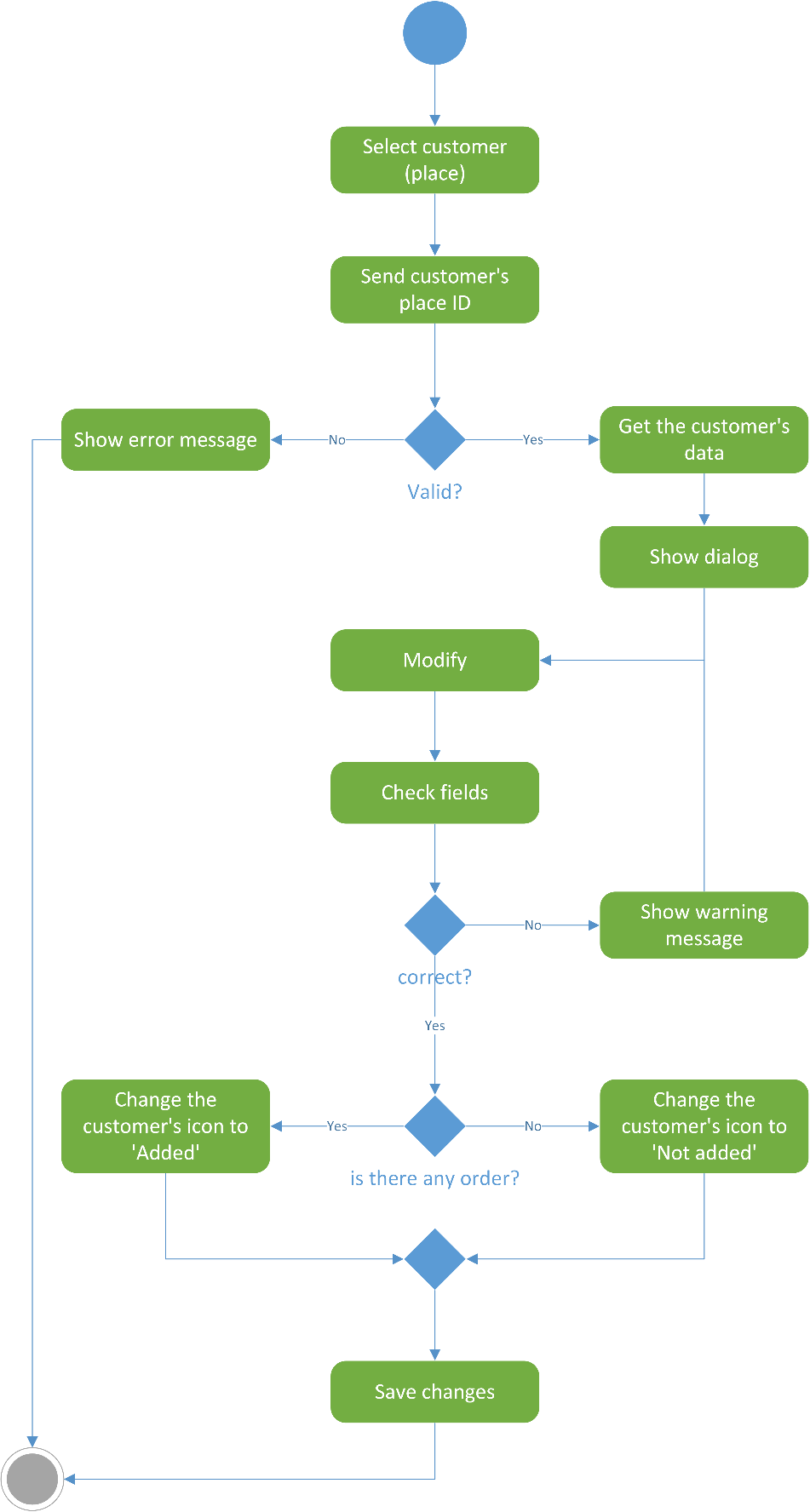


Figure 16 Activity diagram "Edit customer"

### Package diagram

The package diagram for mission management system is shown in figure 17.

A package diagram in the Unified Modeling Language depicts the dependencies between the packages that make up a model. In addition to the standard UML Dependency relationship, there are two special types of dependencies defined between packages:

1. package import
2. package merge

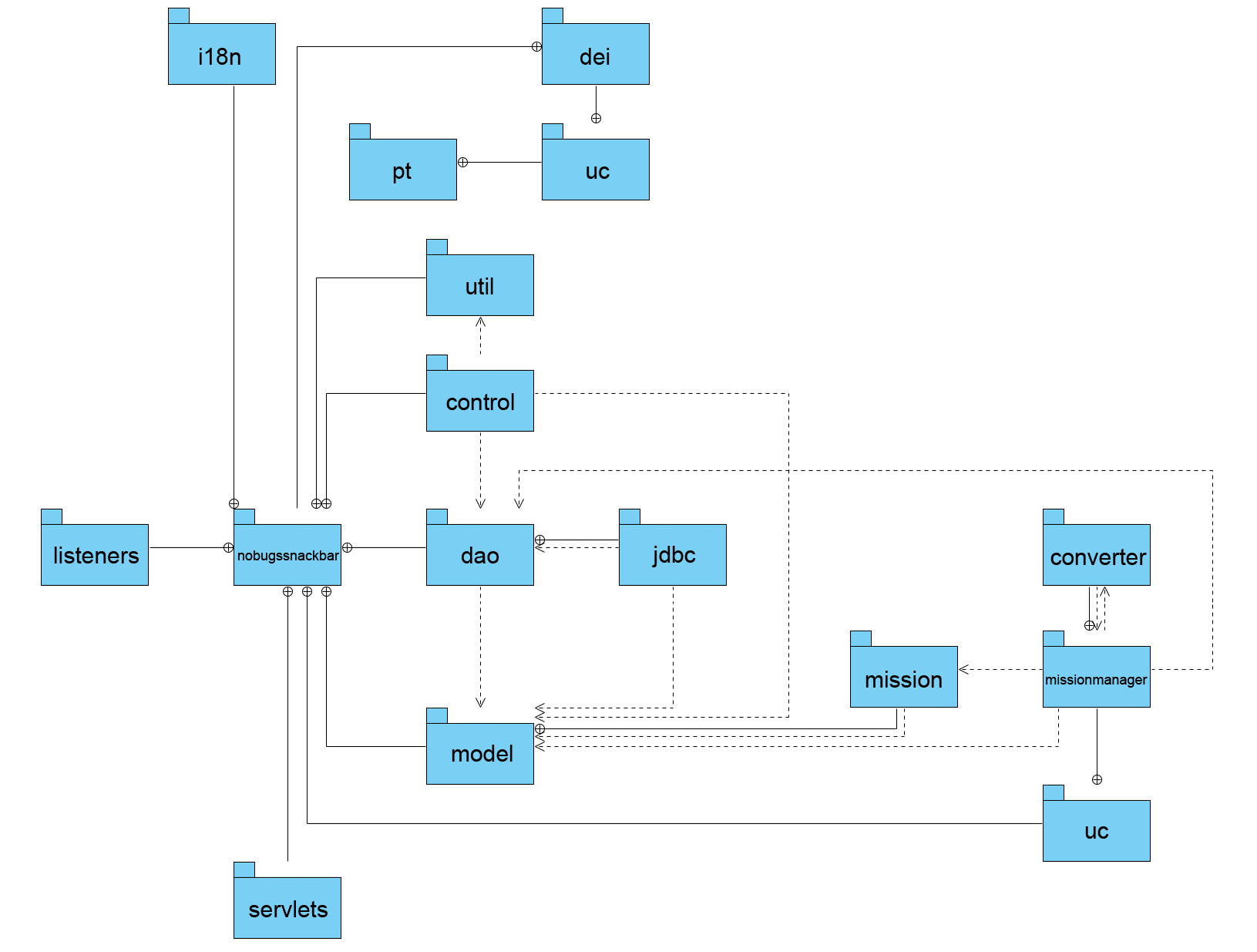


Figure 17 Package diagram of the system

A package import is a relationship between an importing namespace and a package, indicating that the importing namespace adds the names of the members of the package to its own namespace. By default, an unlabeled dependency between two packages is interpreted as a package import relationship. In this relationship, elements within the target package will be imported into the source package.

A package merge is a directed relationship between two packages, which indicates that the contents of the two packages are to be combined. It is very similar to Generalization in the sense that the source element conceptually adds the characteristics of the target element to its own characteristics resulting in an element that combines the characteristics of both In this relationship, if an element exists within both the source package and the target package, then the source element's definition will be expanded to include the target element's definition [7].

### Class diagram

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

In the diagram, classes are represented with boxes which contain three parts:

1. The top part contains the name of the class. It is printed in bold and centered, and the first letter is capitalized;
2. The middle part contains the attributes of the class. They are left-aligned and the first letter is lowercase;
3. The bottom part contains the methods the class can execute. They are also left-aligned and the first letter is lowercase [8].

The general class diagram of this project is too big (unreadable) for borders of this paper and it is skipped, but you can find it in the compaq disk (CD), applied with this paper.

The figures, below, from 18 to 28 are class diagrams of the classes in each package.

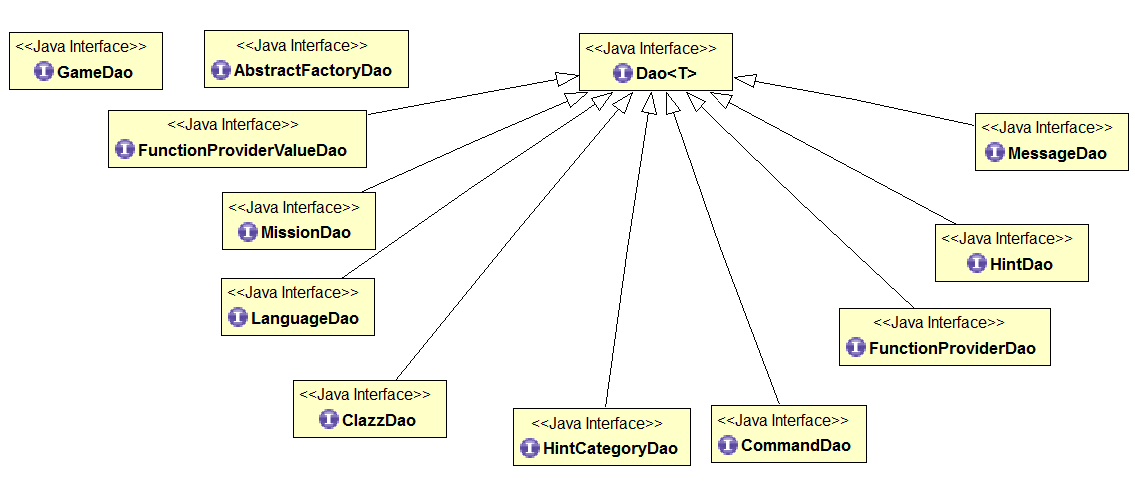


Figure 18 Class diagram of package "dao"

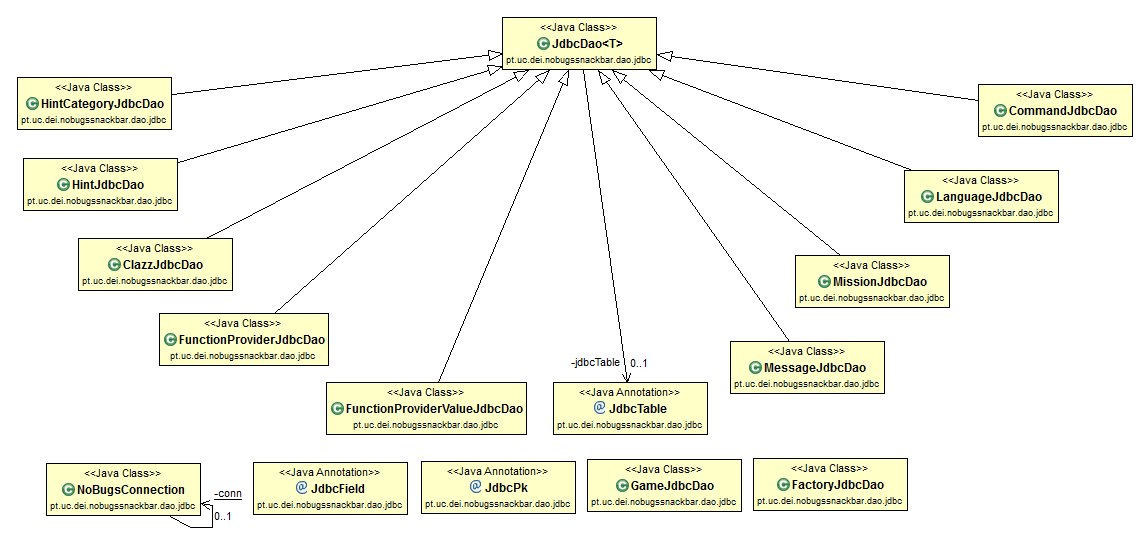


Figure 19 Class diagram of package "jdbc"

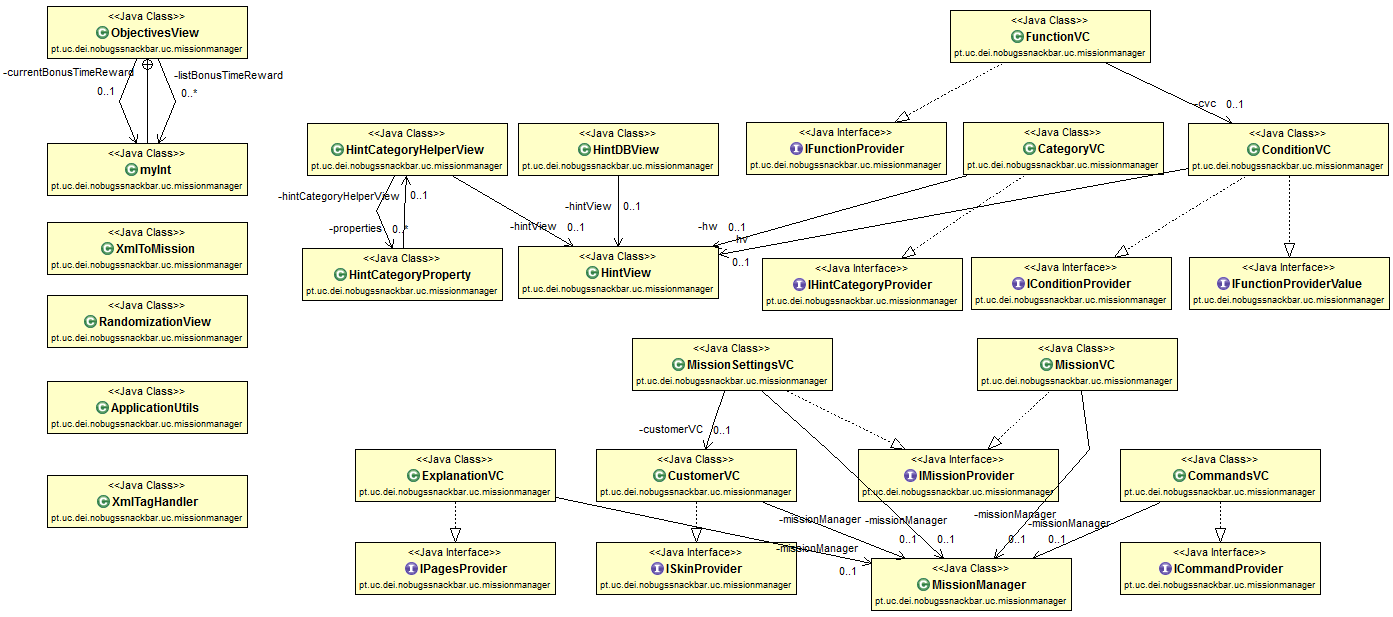


Figure 20 Class diagram of package "missionmanager"

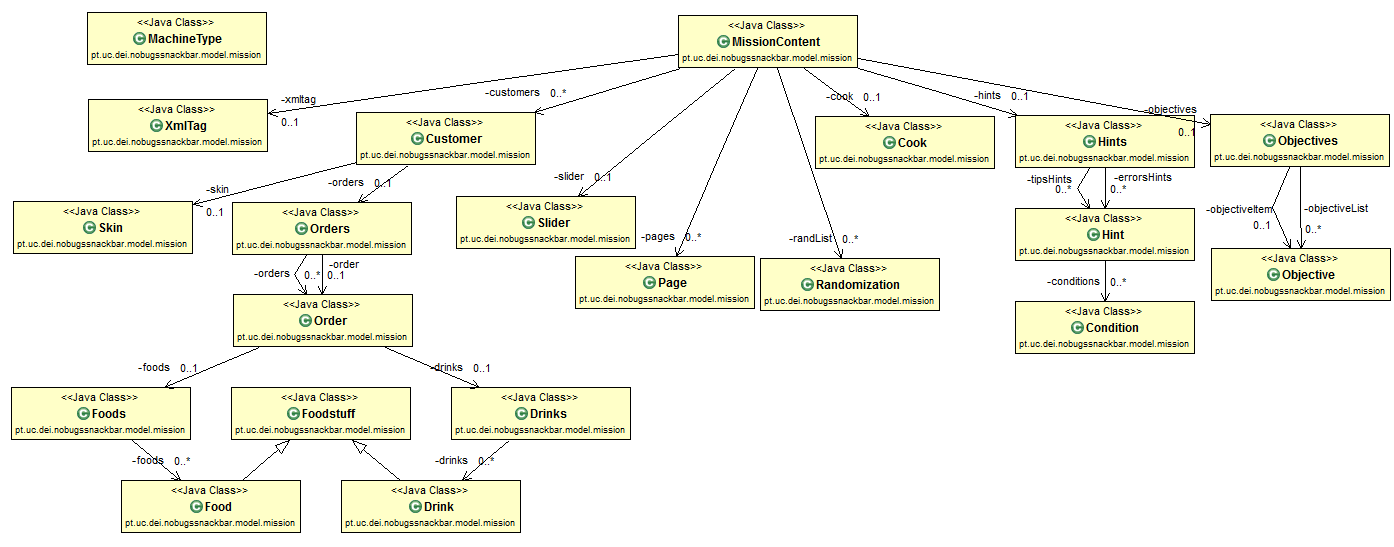


Figure 21 Class diagram of package "mission"

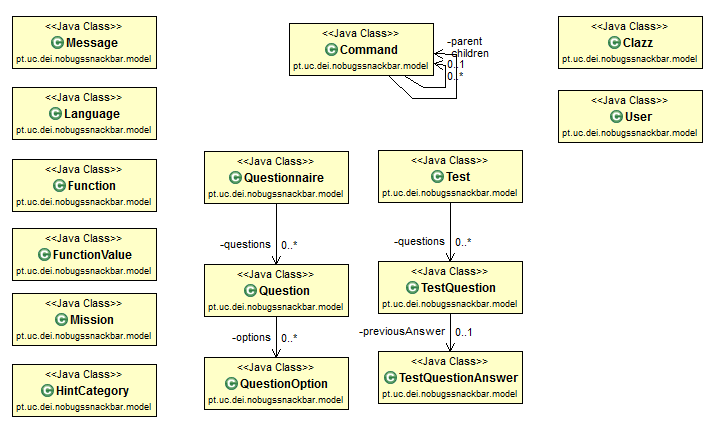


Figure 22 Class diagram of package "model"



Figure 23 Class diagram of package "servlets"

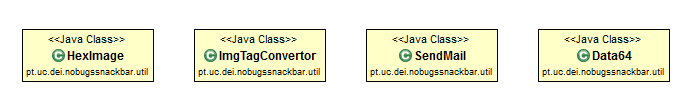


Figure 24 Class diagram of package "util"

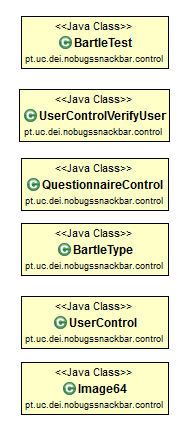


Figure 25 Class diagram of package "control"

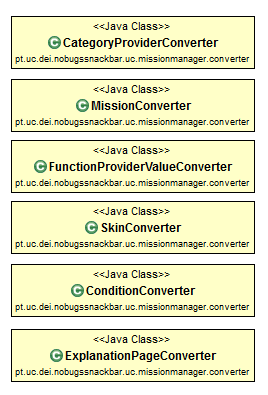


Figure 26 Class diagram of package "converter"



Figure 27 Class diagram of package "listeners"

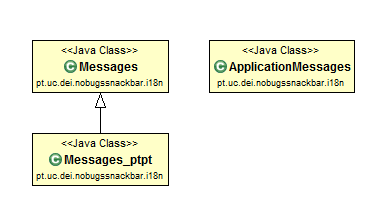


Figure 28 Class diagram of package "i18n"

### Sequence diagram

A sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner [9].

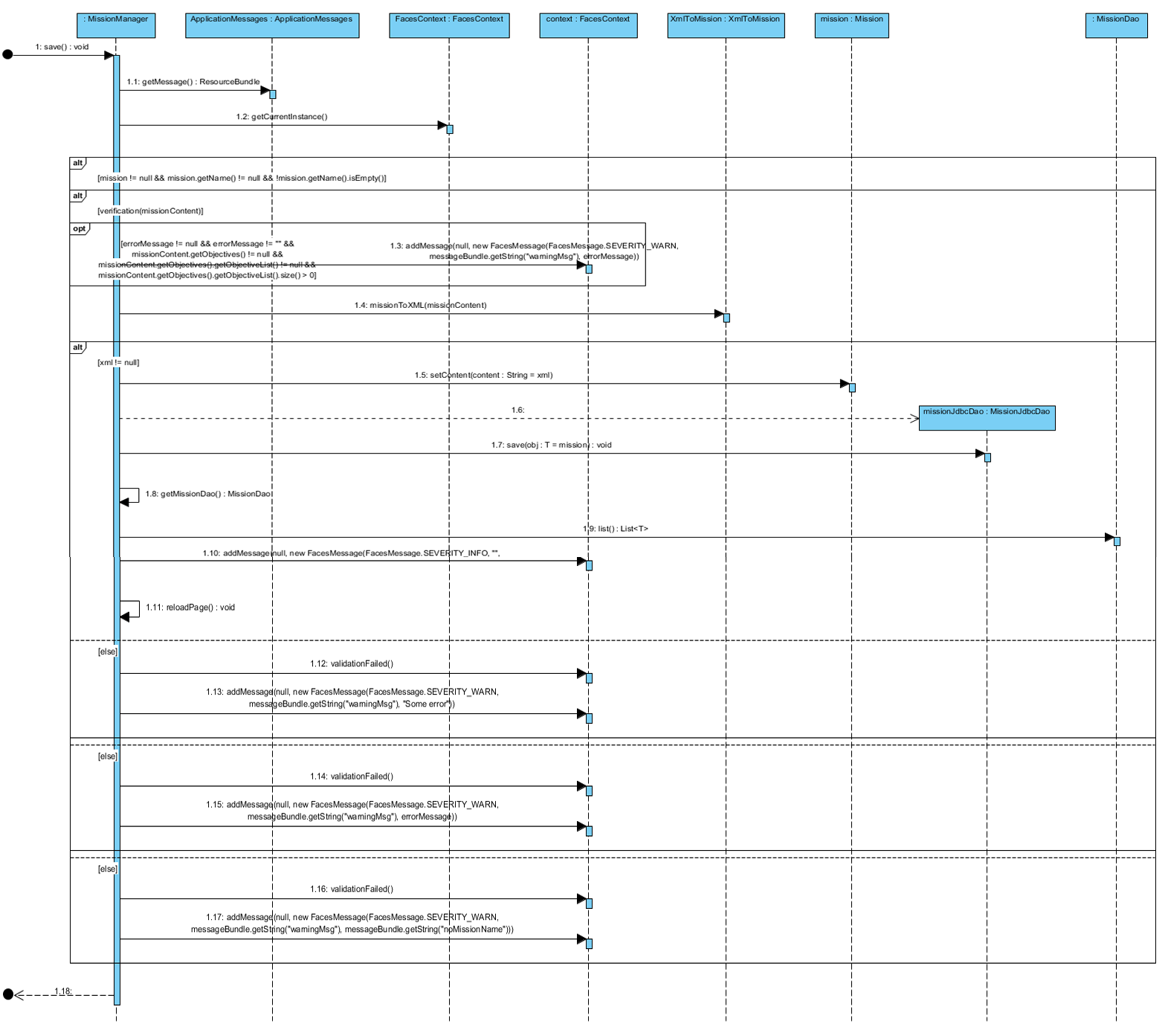


Figure 29 Sequence diagram "Save mission"

In figure 17 is shown a sequence diagram of use case “save mission”.

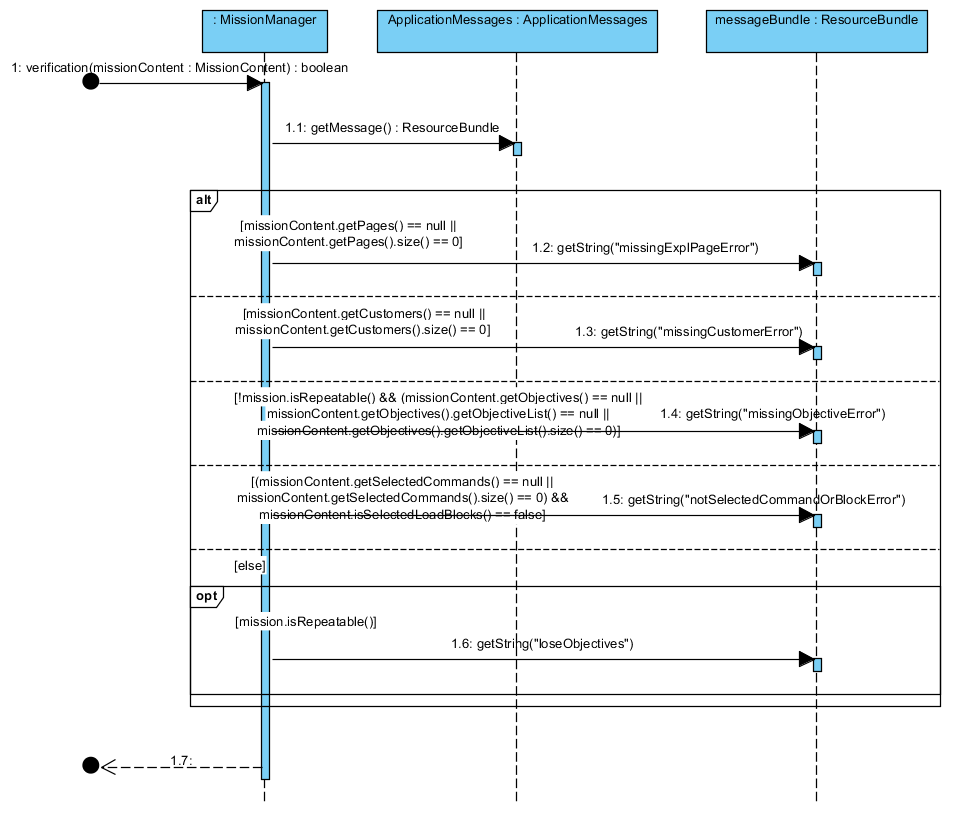


Figure 30 Sequence diagram "Mission verification"

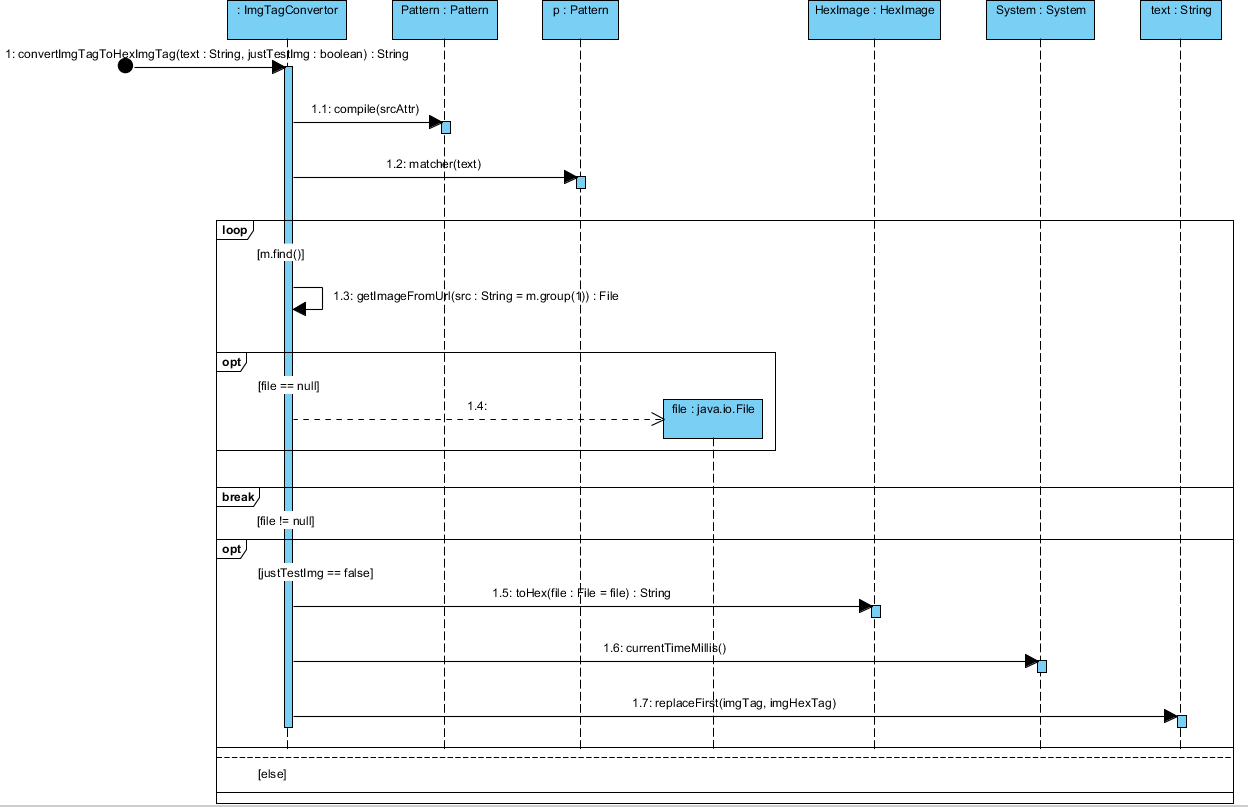


Figure 31 Sequence diagram "Image to "text" conversion"

## System architecture

A system architecture (figure 32) is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

A system architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them [10].

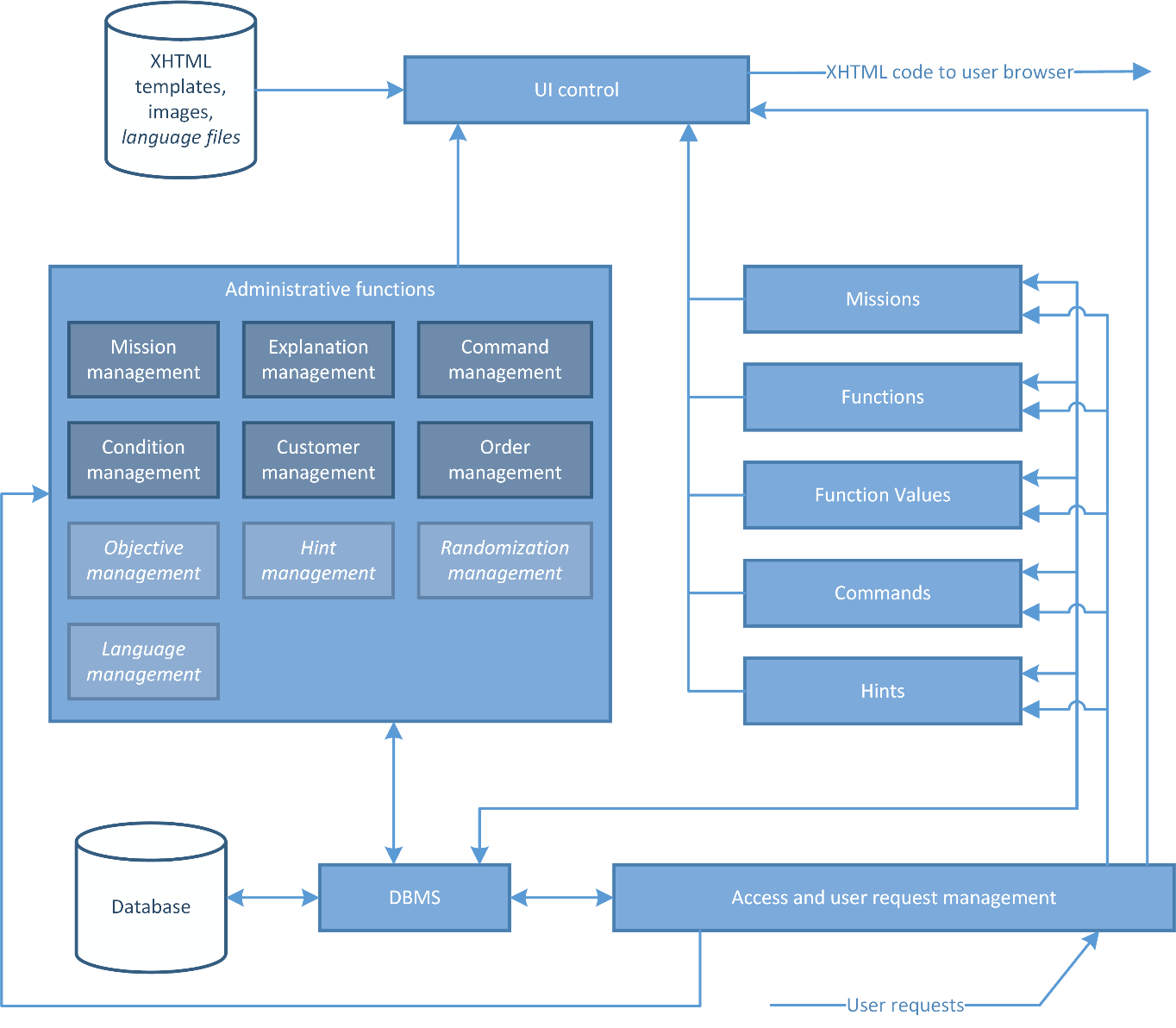


Figure 32 System architecture

## Data organization – conceptual schema

As you know this project is a part of a bigger one – the PhD project “Improving Problem-Solving Skills in an Introductory Computer Programming Learning using a Digital Game” by [Adilson Vahldick](https://www.facebook.com/adilson.vahldick). So, the database was already designed. During the development of this project was necessary to create new tables and change existing ones, according to the needs.

The conceptual schema of database which contains *only* tables (entities with dark background), used in this project is shown in figure 33.

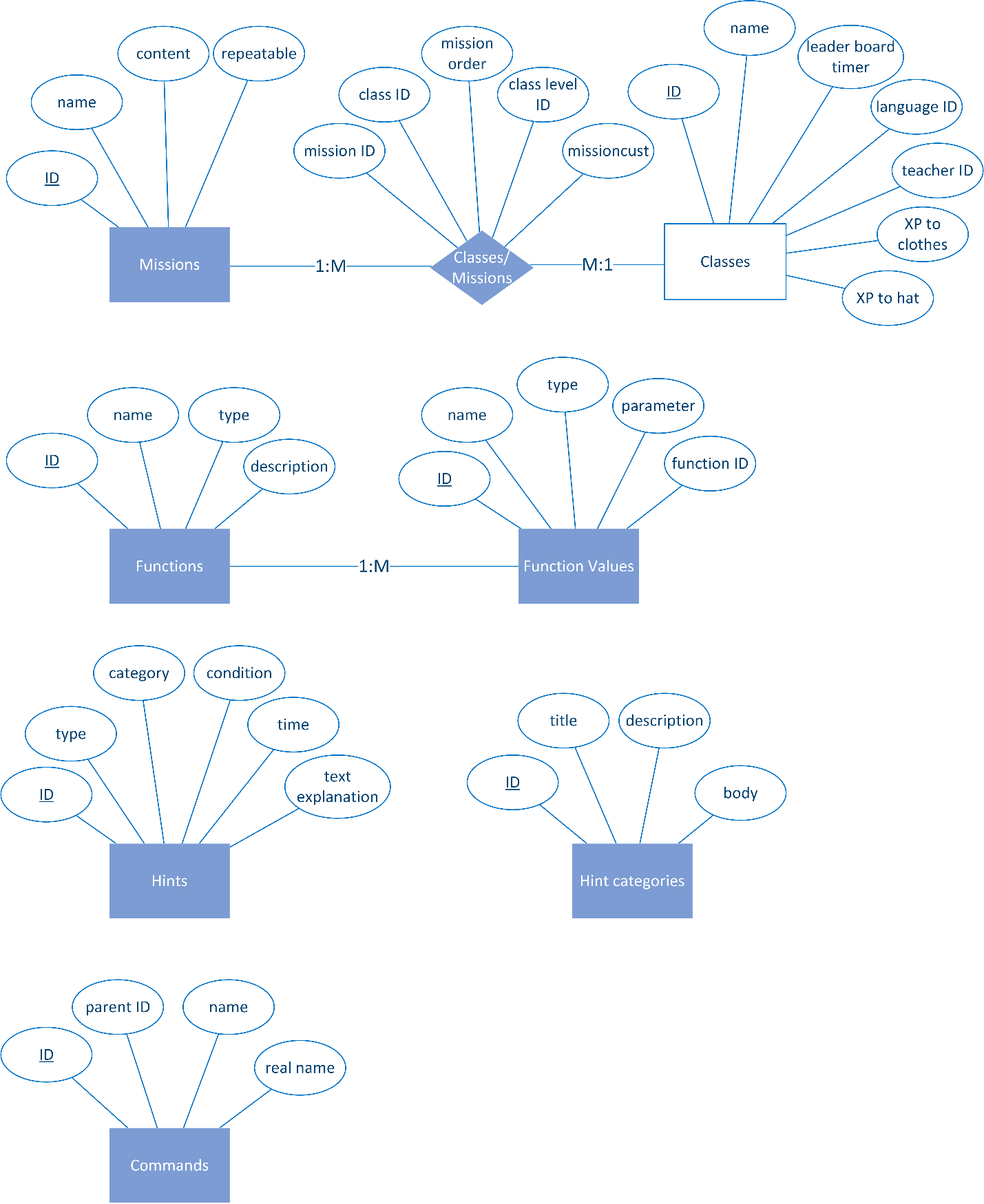


Figure 33 Conceptual schema

A conceptual schema or conceptual data model is a map of concepts and their relationships used for databases. This describes the semantics of an organization and represents a series of assertions about its nature. Specifically, it describes the things of significance to an organization (entity classes), about which it is inclined to collect information, and characteristics of (attributes) and associations between pairs of those things of significance (relationships) [11].

## Implementation

### Data structure

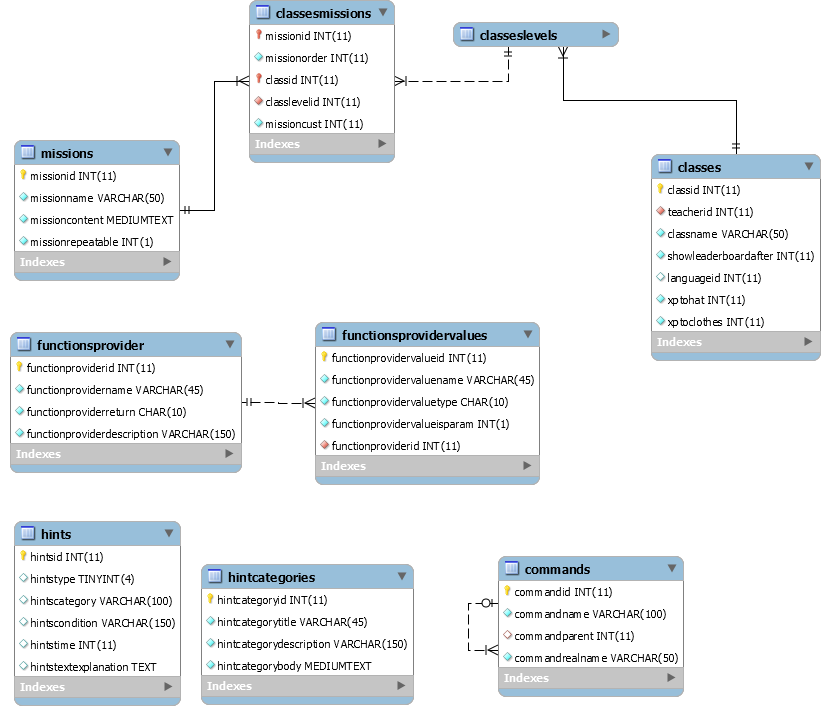


Figure 34 Database schema

Table 1 Table "Mission"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Characteristic | Description |
| missionid | Int(11) | Primary Key (PK), Not Null (NN), Auto Incremental (AI) | Identificator of mission |
| missionname | Varchar(50) | NN | Name of mission |
| missioncontent | Mediumtext | NN | Contains XML code of mission |
| missionrepeatable | Int(1) | NN | Whether mission is repeatable or not |

Table 2 Table "ClassesMissions"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Characteristic | Description |
| missionid | Int(11) | PK, Foreign Key (FK), NN | Identificator of mission |
| missionorder | Int(11) | NN | Mission order |
| classid | Int(11) | PK, FK, NN | Identificator of class |
| classlevelid | Int(11) | FK, NN | Identificator of class level |
| missioncust | Int(11) | NN |  |

Table 3 Table "FunctionProvider"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Characteristic | Description |
| functionproviderid | Int(11) | PK, NN | Identificator of function |
| Functionprovidername | Varchar(45) | NN | Name of function |
| Functionproviderreturn | Char(10) | NN | Return type of function |
| functionproviderdescription | Varchar(150) | NN | Description |

Table 4 Table "FunctionProviderValues"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Characteristic | Description |
| functionprovidervalueid | Int(11) | PK, AI, NN | Identificator |
| functionprovidervaluename | Varchar(45) | NN | Name |
| functionprovidervaluetype | Char(10) | NN | Type |
| functionprovidervalueisparam | Int(1) | NN | Whether the value can be used for parameter too |
| functionproviderid | Int(11) | FK, NN | Which function can use this value |

Table 5 Table "Hints"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Characteristic | Description |
| hintsid | Int(11) | PK, AI, NN | Identificator |
| hintstype | Tinyint(4) | Default Value (DV) - NULL | Type |
| hintscategory | Varchar(100) | DV - NULL | Category |
| hintscondition | Varchar(150) | DV - NULL | Condition |
| hintstime | Int(11) | DV - NULL | Time |
| hintstextexplanation | Text |  | Explanation |

Table 6 Table "HintCategories"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Characteristic | Description |
| hintcategoryid | Int(11) | PK, AI, NN | Identificator |
| hintcategorytitle | Varchar(45) | NN | Title |
| hintcategorydescription | Varchar(150) | NN | Description |
| hintcategorybody | Mediumtext | NN | Body – XML code for auto-generating of dialog |

Table 7 Table "Commands"

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Characteristic | Description |
| commandid | Int(11) | PK, AI, NN | Identificator |
| commandname | Varchar(100) | NN | Name |
| commandparent | Int(11) | FK, DV – NULL | Parent ID |
| commandrealname | Varchar(50) | NN | Real Name |

### Technologies

The technologies involved in this project were Java, Java Server Faces (JSF), PrimeFaces, HTML5 & CSS3, XML and MySQL.

#### programming language and deployment environment

It was required to use the programming language Java and a Java specification for building component-based user interfaces for web applications, called Java Server Faces. So, it was necessary to choose a deployment environment. There are two very popular ones – Eclipse and NetBeans, IntelliJ also. The last one is paid, so it drops out.

**Eclipse**

Eclipse has been in existence from the year 2001, ever since IBM (International Business Machines Corporation) released Eclipse as an open source platform. Managed by the non-profit Eclipse Foundation, this is used in both open source and commercial projects. Starting in a humble manner, this has now emerged as a major platform, which is also used in several other languages.

The greatest advantage of Eclipse is that it features a whole plethora of plugins, which makes it versatile and highly customizable. This platform works for you in the background, compiling code and showing up errors as when they occur. The entire IDE is organized in Perspectives, which are essentially sort of visual containers, which offer a set of views and editors.

Eclipse’s multitasking, filtering and debugging are yet other pluses. Designed to fit the needs of large development projects, it can handle various tasks such as analysis and design, product management, implementation, content development, testing and documentation as well.

**NetBeans**

NetBeans was independently developed in the latter half of the 1990s. It emerged as an open source platform after it was acquired by Sun in 1999. Now a part of Oracle, this IDE can be used to develop software for all versions of Java ranging between Java ME, up to the Enterprise Edition. Like Eclipse, NetBeans too features a variety of plugins you can work with.

NetBeans offers you various different bundles – 2 C/C++ and PHP editions, a Java SE edition, the Java EE edition, and 1 kitchen sink edition that offers everything you will ever need for your project. This IDE also offers tools and editors which can be used for HTML, PHP, XML, JavaScript and more. You can now find support for HTML5 and other Web technologies as well.

NetBeans scores over Eclipse in that it features database support, with drivers for Java DB, MySQL, PostgreSQL and Oracle. Its Database Explorer enables you easily create, modify and delete tables and databases within the IDE.

Largely viewed in the past as a sort of shadow of Eclipse, NetBeans has now emerged as a formidable competitor to the former [12].

**In Conclusion**

**All of the above integrated development environments (IDEs) come with their own advantages. While Eclipse is still the widest used IDE, NetBeans is now gaining popularity with independent developers.**

#### **database management system**

There are a lot of database management system (DBMS). Well-known DBMSs include MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase and IBM DB2 [13].

The choice for DBMS was taken by project manager's advice – MySQL. MySQL is the world’s most popular open source database, enabling the cost-effective delivery of reliable, high-performance and scalable Web-based and embedded database applications, including all five of the top five websites [14, 15].

#### Component library for jsf

The most popular JSF component libraries are RichFaces, IceFaces and PrimeFaces. According to [DevRates.com](http://devrates.com/stats/index), PrimeFaces has been the second (at time of writing) developers’ favorite framework to create rich user interfaces with Java, while RichFaces has been on the last place (figure 35). An interesting news is that PrimeFaces argues that IceFaces is powered by them [17].

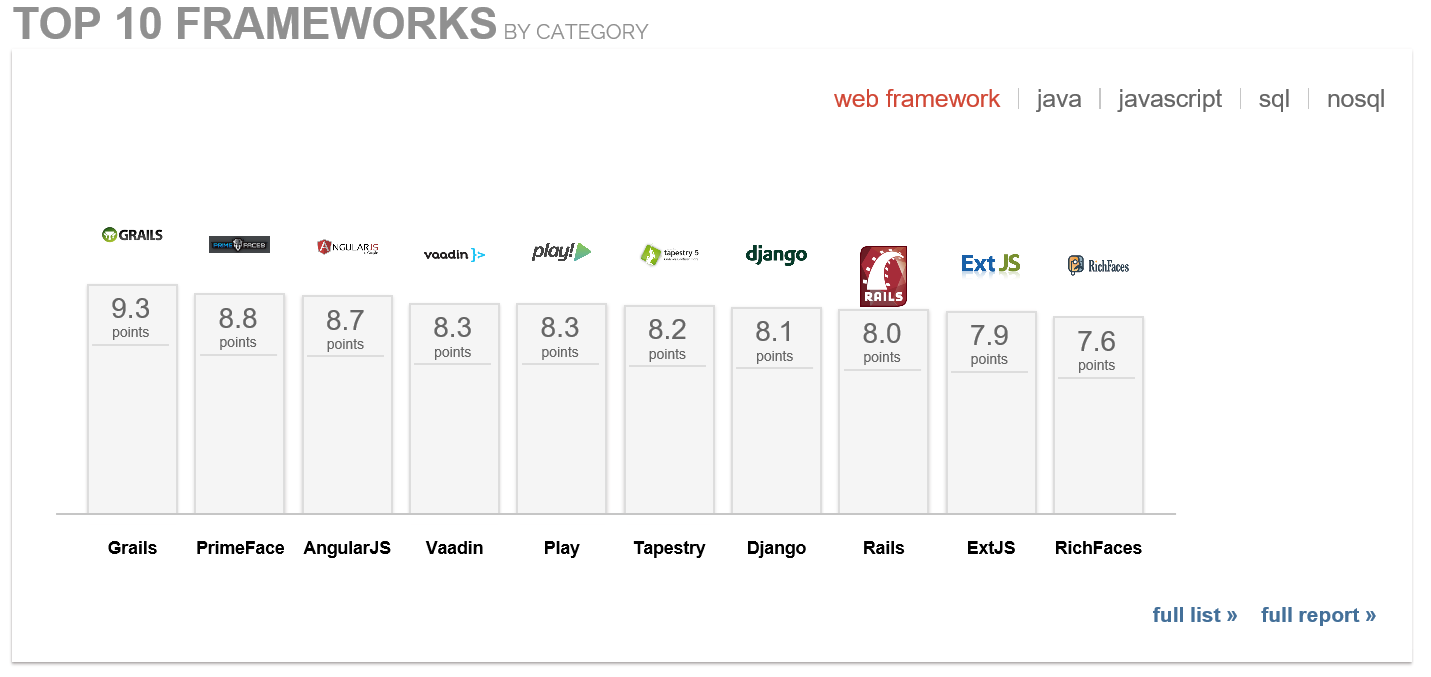


Figure 35 Top 10 Web frameworks [16]

Figure 36 is generated by [Google Trends](https://www.google.com/trends/) comparing the popularity of PrimeFaces with the competitors. Those facts are enough to choose PrimeFaces.

PrimeFaces is a component suite open source UI component library for JSF based applications, created by PrimeTek, Turkey [18]. It offers:

Rich set of components (HtmlEditor, Dialog, AutoComplete, Charts and many more).

* Built-in Ajax based on standard JSF 2.0 Ajax application programming interfaces (APIs);
* Lightweight, one jar, zero-configuration and no required dependencies;
* Ajax Push support via websockets;
* Mobile UI kit to create mobile web applications for handheld devices;
* Skinning Framework with 30 built-in themes and support for visual theme designer tool;
* Extensive documentation;
* Large, vibrant and active user community [19].

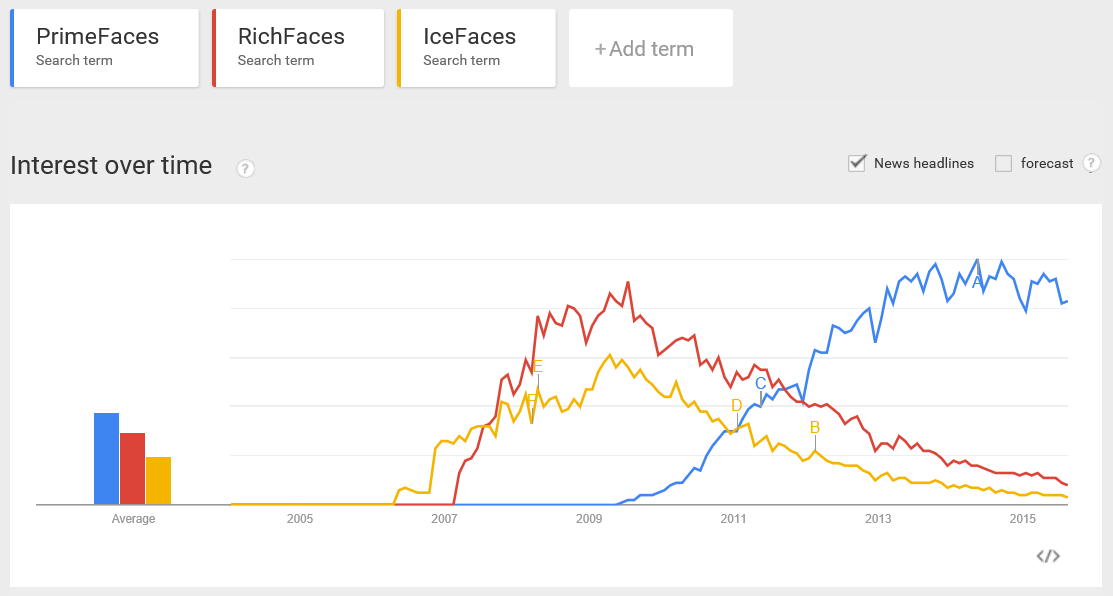


Figure 36 JSF Component libraries - popularity comparison

#### Web-based Git repository hosting service

This project, as it was mentioned, is a part of other one and in order to facilitate sharing of changes, made during the development, was used web-based git repository hosting service.

Initially, google code was providing this service, but after announcing by google that [google code is shutting down](http://google-opensource.blogspot.com/2015/03/farewell-to-google-code.html) the project was moved on GitHub.

GitHub is a Web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a Web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project [20].

### Description of main modules

#### module “mission management”

* **Internal structure**
  + Sub-module for managing of missions;
  + Sub-module for converting a mission to XML code;
  + Sub-module for managing attributes of mission;
* **Sub-module for managing of missions**
  + **Description**

It implements logic for mission selection, deletion, adding, updating and loading of data from DB. Also, it validates the data before doing any of the mentioned operations.

* + **Depends on**

Modules for managing: attributes of mission, explanations, commands, customers, randomization, objectives and hints, sub-module for converting a mission to XML code, and DBMS.

* + **Implementation**
    - **public void save()** – it validates data, calls module for marshalling (converting) the mission object to XML code and then saves/updates the mission in DB;
    - **public void load()** – for loading a mission from DB. Firstly, the mission is unmarshalled to a mission object;
    - **public void deleteMission()** – removes a mission from DB;
    - **public boolean verification(MissionContent missionContent)** – checks if all required information is okay;
    - **public void checkIfSelected()** – checks if it is selected a mission;
    - **public void handleMissionSelect(SelectEvent event)** – handles the event when a mission is selected;
    - **static void reloadPage()** – refreshes the UI.
* **Sub-module for converting a mission to XML code**
  + **Description**

It is used for converting a mission object to XML code.

* + **Depends on nothing**
  + **Implementation**
    - **public static String missionToXML(MissionContent mc)** – checks the data and marshals it to XML code;
* **Sub-module for managing attributes of mission (mission settings)**
  + **Description**

It is used for managing attributes of mission.

* + **Depends on**

Sub-module for managing of missions and module for managing of blocks for visual programming (not a direct part of this project).

* + **Implementation**
    - **private void init()** – post-constructor;
    - **private Mission selectMission(Integer id)** – returns the mission, loaded from DB, with identificator equals to id;
    - **This module includes many getters and setters**. **There is not a strong business logic behind it.**

#### module “management of explanations”

* **Internal structure**
  + Sub-module for managing of operations on explanations;
  + Sub-module for checking of images;
* **Sub-module for managing of operations on explanations**
  + **Description**

It implements functions as adding, deleting, editing and reordering (and data validation).

* + **Depends on**

Sub-module for checking of images and module “mission management”

* + **Implementation**
    - **public void addPage()** – for adding an explanation page into the list of the explanation pages;
    - **public void editPage()** – for updating an explanation page which is in the list;
    - **public void getPageById()** – finds the “page” by its ID;
    - **public void deleteExplPage()** – deletes a “page” from the list;
    - **private int indexOfPageById(int id, List<Page> list)** – returns the index of the “page” by its ID.
* **Sub-module for checking of images**
  + **Description**

It is used for checking (whether their URL addresses and types are correct) and converting the images represented in “explanation pages” as “img” tags into “imghex”;

* + **Depends on**

Module for converting of image to/from sequence of hexadecimal numbers;

* + **Implementation**
    - **public static String convertImgTagToHexImgTag(String text, boolean justTestImg)** – if “justTestImg” is “true” it’ll check the image tags’ URLs whether they work and are in correct format. Otherwise, it’ll do the same but in addition it’ll call the module for converting of image to sequence of hexadecimal numbers and replace the “img” tags with “imghex”.

#### Module “management of commands”

* **Description**

It implements logic for loading the commands from DB, dividing them by their type and select/deselect function.

* **Depends on**

Module “Mission Management”.

* **Implementation**
  + **private void init()** – post-constructor. It loads the commands using the module “Mission Management” and does some changes with them;
  + **public void handleRootSelect(Command item)** – sets root command’s value to the child items;
  + **public void handleChildSelect(Command item)** – sets root command’s value as “selected” if there is even one selected child item;
  + **public List<Command> getSelectedCommands()** – returns a list with all selected commands (roots + childs).

#### module “managing of customers”

* **Internal structure**
  + Sub-module for managing customer’s settings;
  + Sub-module for managing of orders.
* **Sub-module for managing customer’s settings**
  + **Description**

It is used for managing customer’s settings. Such as customer’s skin, initial and destination positions.

* + **Depends on**

Image files located in the project directory and sub-module for managing of orders.

* + **Implementation**
    - **public void handleCustomer(ActionEvent event)** – gets the selected customer’s data from the customer list;
    - **public void saveCustomer()** – updates customer’s data;
    - **private void initMainLists()** – initializes the initial and destination positions list.
* **Sub-module for managing of orders**
  + **Description**

It executes operations as “add order”, “delete order”, “reorder order list”, “add/edit/delete foodstuff or beverage”.

* + **Depends on nothing**
  + **Implementation**
    - **public void newOrder()** – for creating a new “empty” order but also saves changes in the previous order;
    - **public void deleteOrder()** – for removing an order from the order list;
    - **private boolean checkFields(Order order)** – for checking if the data is filled out correctly;
    - **public void addFoodstuff()** – for adding a foodstuff or a drink into the foods/drinks list of order;
    - **public void deleteFood()** – for removing of a food;
    - **public void deleteDrink()** – for removing of a drink.

#### module “condition builder”

* **Internal structure**
  + Sub-module for managing of functions;
  + Sub-module for managing of condition’s elements;
* **Sub-module for managing of functions**
  + **Description**

This module loads the functions (and their specific values), available in the mission, from the database. It filters functions by key words.

* + **Depends on**

DBMS.

* + **Implementation**
    - **public void handleKeyUpEvent()** – for filtering functions by key word;
    - **public void getFunctionById()** – for searching a function in DB by ID. It also manages function value list according to the selected function.
* **Sub-module for managing of condition’s elements**
  + **Description**

It manages the logic for creating/editing/validating of a “condition”.

* + **Depends on**

Sub-module for managing of functions.

* + **Implementation**
    - **public void newOrEditCondList()** – if the selected condition is not created yet it will create an empty list. However if it is, it will load it.
    - **private int indexOfConditionById(long id, List<Condition> list)** – returns the index of the condition by its ID;
    - **public Condition getConditionById()** – returns the condition by its ID;
    - **public void addCondition(boolean logicalOperator)** – for adding a logical operator (“and” / ”or”) into the condition list;
    - **public void addCondition()** – for adding a sub-condition into the list;
    - **public void deleteCondition()** – for removing a condition;
    - **public void editCondition()** – loads a sub-condition for editing;
    - **public boolean checkCondition()** – checks if the elements of the condition list are in correct order;
    - **public boolean checkFields()** – checks if fields of sub-condition are filled out correctly.

### Structure and organization of user interface

In figure 37 is shown the structure of user interface. Logically, it is divided into four sections. The first section, called “Section A”, contains two controls: text field for name of the mission and drop down menu for choosing language of UI.

The second section “Section B” is the main content. It has two sub-sections: navigation menu and content.

“Section C” contains only two buttons. A button for saving mission and an other one for testing mission.

“Section D” contains a list with available mission in the game.

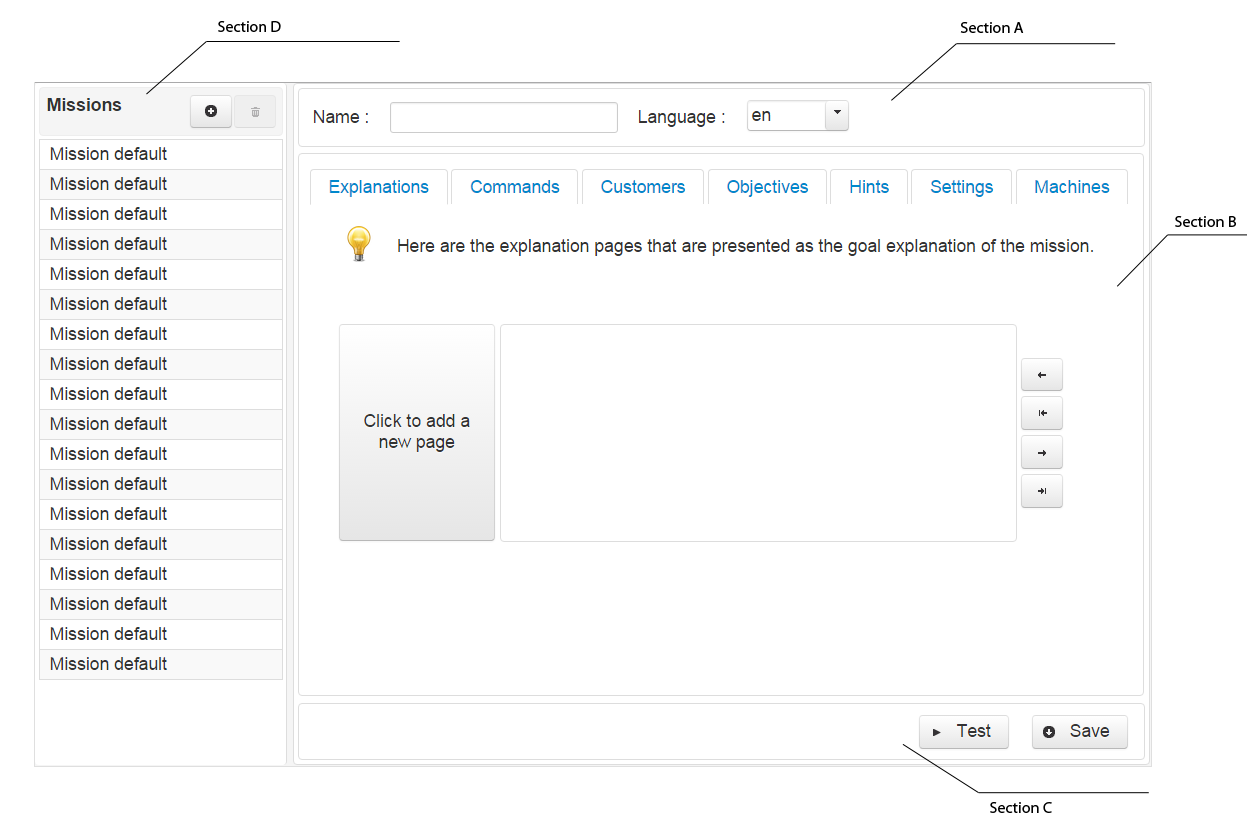


Figure 37 Structure of UI

Actually, the application is a single page web application. So, there is only one page and instead of using different pages – dialogs are used.

The sitemap is showing the dialogs in the application is in figure 38.

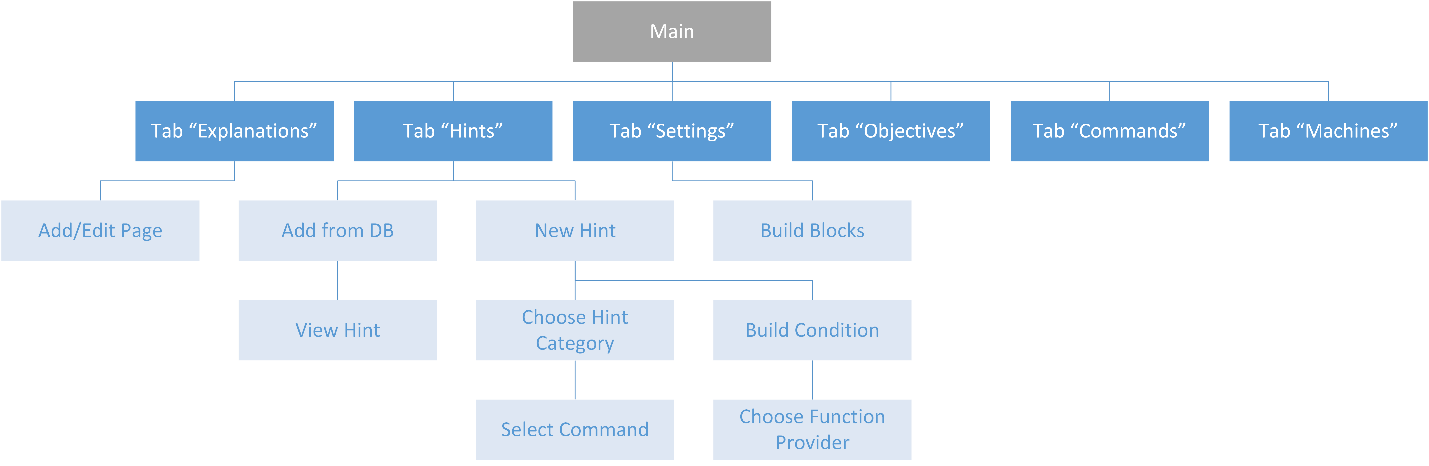


Figure 38 Sitemap

### Format of input and output documents

As input can be used both XML code (for editing – it will be unmarshalled) and user interface. The output is always XML code, which has the same format as the input.

Below is a sample code of mission:

<?xml version=*'1.0'*?>

<mission open=*"true"* timeLimit=*"600"*>

<explanation hasInstruction=*"false"*>

<page type=*"goal"*>

<![CDATA[

Chegamos a um tipo diferente de miss&#227;o. Neste tipo de miss&#227;o voc&#234; pode rejogar v&#225;rias

vezes, atrav&#233;s de um custo de moedas. <br/>

<br/>

Entretanto, voc&#234; poder&#225; melhorar a sua esplanada atrav&#233;s da compra de m&#225;quinas e

equipamentos. Para isso, basta clicar no bot&#227;o<imghex id="m17\_iimghex>.<br/>

<br/>

Na janela que se abrir&#225; consegues selecionar que equipamento queres instalar, de acordo com a tua

disponibilidade de dinheiro. A partir da&#237;, ser&#227;o disponibilizados novos comandos para usar o

equipamento. Esses comandos estar&#227;o acess&#237;veis a partir do menu [Seus equipamentos].

]]>

</page>

<page type=*"goal"*>

<![CDATA[

Voc&#234; tem um tempo limite para fazer dinheiro. Os clientes v&#227;o entrar e sair do seu estabelecimento.

Tome cuidado para respeitar o tempo de entrada e sa&#237;da dos clientes. Lembre-se: se perguntar a um espa&#231;o

vazio o que ele deseja comer, o programa p&#225;ra a execu&#231;&#227;o devido a um erro. <br/><br/>

Observe primeiro o padr&#227;o de pedidos dos clientes. Os clientes agora tem um tempo limite para serem atendidos e

se ir&#227;o embora se n&#227;o forem atendidos dentro desse tempo. Ent&#227;o observe que as vontades deles, apesar de

mudarem, seguem um padr&#227;o.<br/><br/>

A execu&#231;&#227;o do seu programa se repetir&#225; automaticamente at&#233; encerrar o tempo.

]]>

</page>

<page type=*"goal"*>

<![CDATA[

Essas miss&#245;es poder&#227;o lhe dar b&#244;nus quanto <b>menos vezes</b> voc&#234; clicar em executar ou menos

c&#243;digo tiver para atender os pedidos. <br/><br/>

Mas aten&#231;&#227;o: sempre que a execu&#231;&#227;o parar por causa de um erro, voc&#234; deixa de ganhar todas

as moedas que tinhas coletado at&#233; aquele instante.

]]>

</page>

</explanation>

<hints>

<sequence>

</sequence>

<!-- <errors> only happens after running a program, and some goals aren't achieved -->

<errors>

<hint category=*"LastError"* condition=*"Hints.lastErrorId() == 'Error.thereIsntCustomer'"*>

<![CDATA[

Observe que o seu cozinheiro chegou antes do cliente. Que tal fazer o seu cozinheiro gastar algum tempo antes de alcan&#231;ar esse s&#237;tio ?

]]>

</hint>

</errors>

</hints>

<selectMachine>

<machine type=*"1"*/> <!-- ice cream -->

<machine type=*"2"*/> <!-- french fryer -->

</selectMachine>

<commands>

<category name=*"snackMan"* show=*"true"*/>

<category name=*"goToBarCounter"* show=*"true"*/>

<category name=*"askForFood"* show=*"true"* />

<category name=*"goToDisplay"* show=*"true"* />

<category name=*"pickUpHotDog"* show=*"true"* />

<category name=*"deliver"* show=*"true"* />

<category name=*"wait"* show=*"true"*/>

<category name=*"vars"* show=*"true"* />

</commands>

<cooker>initial</cooker>

<customers>

<customer>

<id>01</id>

<init>counter1</init>

<dest>counter1</dest>

<orders>

<order>

<foods>

<food price=*"1"*>hotdog</food>

</foods>

<drinks>

</drinks>

</order>

<order>

<foods>

<food price=*"1"*>machine1-prod</food>

<food price=*"1"*>hotdog</food>

</foods>

<drinks></drinks>

</order>

</orders>

</customer>

<customer>

<id>02</id>

<init>counter3</init>

<dest>counter3</dest>

<orders>

<order>

<foods>

<food price=*"1"*>hotdog</food>

</foods>

<drinks></drinks>

</order>

<order>

<foods>

<food price=*"1"*>machine2-prod</food>

</foods>

<drinks></drinks>

</order>

</orders>

</customer>

</customers>

<objectives buttonBuy=*"true"*/>

<xml alwaysNew = *"true"*>

<block type=*"do\_wait"*>

<value name=*"VALUE"*>

<block type=*"math\_number"*>

<field name=*"NUM"*>5</field>

</block>

</value>

</block>

</xml>

</mission>

This code is generated according to the XML Schema Definition (XSD) created for the purposes of this project. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed.

XSD is a World Wide Web Consortium (W3C) recommendation that specifies how to formally describe the elements in an XML document. In general, a schema is an abstract representation of an object's characteristics and relationship to other objects. An XML schema represents the interrelationship between the attributes and elements of an XML object (for example, a document or a portion of a document). The process of creating a schema for a document involves analyzing its structure and defining each structural element encountered. For example, a schema for a document describing a website would define a website element, a webpage element, and other elements that describe possible content divisions within any page on that site. Just as in XML and HTML, elements are defined within a set of tags.

XSD has several advantages over earlier XML schema languages, such as Document Type Definition (DTD) or Simple Object XML (SOX). XSD is written in XML, which means that it doesn't require intermediary processing by a parser. Other benefits include self-documentation, automatic schema creation and the ability to be queried through XML Transformations (XSLT).

There are many challenges and limitations with XSD as well. Some detractors have argued it is unnecessarily complex, lacks a formal mathematical description and has limited support for unordered content [21].

XSD

<?xml version=*"1.0"*?>

<xs:schema xmlns:xs=*"http://www.w3.org/2001/XMLSchema"*

targetNamespace=*"http://nobugssnackbar.dei.uc.pt/xml/ns/mission"*

xmlns=*"http://nobugssnackbar.dei.uc.pt/xml/ns/mission"*

attributeFormDefault=*"unqualified"* elementFormDefault=*"qualified"*>

<xs:element name=*"mission"* type=*"missionType"*/>

<xs:complexType name=*"sliderType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:int"* name=*"timesBefore"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"pageType"* mixed=*"true"*>

<xs:choice maxOccurs=*"unbounded"* minOccurs=*"0"*>

<xs:element type=*"xs:string"* name=*"br"* maxOccurs=*"unbounded"*

minOccurs=*"0"* />

<xs:element type=*"xs:string"* name=*"b"* maxOccurs=*"unbounded"*

minOccurs=*"0"* />

<xs:element type=*"spanType"* name=*"span"* maxOccurs=*"unbounded"*

minOccurs=*"0"* />

<xs:element type=*"imghexType"* name=*"imghex"*

maxOccurs=*"unbounded"* minOccurs=*"0"* />

</xs:choice>

<xs:attribute type=*"xs:string"* name=*"type"*/>

</xs:complexType>

<xs:complexType name=*"spanType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:string"* name=*"style"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"explanationType"*>

<xs:sequence>

<xs:element type=*"pageType"* name=*"page"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"hintType"* mixed=*"true"*>

<xs:sequence>

<xs:element type=*"xs:string"* name=*"b"* maxOccurs=*"unbounded"* minOccurs=*"0"* />

<xs:element type=*"xs:string"* name=*"br"* maxOccurs=*"unbounded"* minOccurs=*"0"* />

<xs:element type=*"imghexType"* name=*"imghex"* maxOccurs=*"unbounded"* minOccurs=*"0"* />

</xs:sequence>

<xs:attribute type=*"xs:string"* name=*"category"* use=*"optional"*/>

<xs:attribute type=*"xs:short"* name=*"time"* use=*"optional"*/>

<xs:attribute type=*"xs:string"* name=*"condition"* use=*"optional"*/>

</xs:complexType>

<xs:complexType name=*"sequenceType"*>

<xs:sequence>

<xs:element type=*"hintType"* name=*"hint"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"imghexType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:string"* name=*"id"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"errorsType"*>

<xs:sequence>

<xs:element type=*"hintType"* name=*"hint"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"hintsType"*>

<xs:sequence>

<xs:element type=*"sequenceType"* name=*"sequence"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"errorsType"* name=*"errors"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"categoryType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:string"* name=*"name"* use=*"optional"*/>

<xs:attribute type=*"xs:string"* name=*"show"* use=*"optional"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"commandsType"*>

<xs:sequence>

<xs:element type=*"categoryType"* name=*"category"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"customerType"*>

<xs:sequence>

<xs:element type=*"xs:integer"* name=*"id"* />

<xs:element type=*"xs:string"* name=*"init"* />

<xs:element type=*"xs:string"* name=*"dest"* />

<xs:element name=*"orders"* type=*"OrdersType"*></xs:element>

</xs:sequence>

<xs:attribute type=*"xs:string"* name=*"randomType"* use=*"optional"*/>

<xs:attribute type=*"xs:integer"* name=*"randomMin"* use=*"optional"* />

</xs:complexType>

<xs:complexType name=*"customersType"*>

<xs:sequence>

<xs:element type=*"customerType"* name=*"customer"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"objectiveType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:integer"* name=*"pos"* use=*"optional"*/>

<xs:attribute type=*"xs:string"* name=*"place"* use=*"optional"*/>

<xs:attribute type=*"xs:boolean"* name=*"distinct"* use=*"optional"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"objectivesType"*>

<xs:sequence>

<xs:element type=*"objectiveType"* name=*"objective"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

<xs:attribute type=*"xs:boolean"* name=*"ordered"*/>

<xs:attribute type=*"xs:integer"* name=*"reward"*/>

<xs:attribute type=*"xs:boolean"* name=*"buttonDebug"*/>

<xs:attribute type=*"xs:boolean"* name=*"buttonRun"*/>

<xs:attribute type=*"xs:boolean"* name=*"variableWindow"*/>

<xs:attribute type=*"xs:boolean"* name=*"buttonBuy"*/>

<xs:attribute type=*"xs:integer"* name=*"commQtd"*/>

<xs:attribute type=*"xs:integer"* name=*"maxCommands"*/>

<xs:attribute type=*"xs:integer"* name=*"maxCommandsReward"*/>

<xs:attribute type=*"xs:integer"* name=*"bonusTime"*/>

<xs:attribute type=*"xs:string"* name=*"bonusTimeReward"*/>

<xs:attribute type=*"xs:integer"* name=*"varQtd"*/>

</xs:complexType>

<xs:complexType name=*"xmlType"*>

<xs:sequence>

<xs:element type=*"blockType"* name=*"block"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

<xs:attribute type=*"xs:boolean"* name=*"alwaysNew"*/>

<xs:attribute type=*"xs:integer"* name=*"preload"*/>

<xs:attribute type=*"xs:integer"* name=*"id"*/>

</xs:complexType>

<xs:complexType name=*"valueType"*>

<xs:sequence>

<xs:element type=*"blockType"* name=*"block"*/>

</xs:sequence>

<xs:attribute type=*"xs:string"* name=*"name"* use=*"optional"*/>

</xs:complexType>

<xs:complexType name=*"nextType"*>

<xs:sequence>

<xs:element type=*"blockType"* name=*"block"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"fieldType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:string"* name=*"name"* use=*"optional"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"blockType"*>

<xs:sequence>

<xs:element type=*"fieldType"* name=*"field"* minOccurs=*"0"*/>

<xs:element type=*"valueType"* name=*"value"* minOccurs=*"0"*/>

<xs:element type=*"nextType"* name=*"next"* minOccurs=*"0"*/>

<xs:element type=*"statementType"* name=*"statement"* minOccurs=*"0"*/>

</xs:sequence>

<xs:attribute type=*"xs:string"* name=*"type"* use=*"optional"*/>

<xs:attribute type=*"xs:integer"* name=*"x"* use=*"optional"*/>

<xs:attribute type=*"xs:integer"* name=*"y"* use=*"optional"*/>

<xs:attribute type=*"xs:boolean"* name=*"inline"* use=*"optional"*/>

<xs:attribute type=*"xs:integer"* name=*"id"*/>

</xs:complexType>

<xs:complexType name=*"wishType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:integer"* name=*"qt"*/>

<xs:attribute type=*"xs:decimal"* name=*"price"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"wishesType"*>

<xs:sequence>

<xs:element type=*"wishType"* name=*"food"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

<xs:attribute type=*"xs:integer"* name=*"randomMin"* use=*"optional"*/>

<xs:attribute type=*"xs:integer"* name=*"randomMax"* use=*"optional"*/>

</xs:complexType>

<xs:complexType name=*"machineType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:integer"* name=*"type"* use=*"optional"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"selectMachineType"*>

<xs:sequence>

<xs:element type=*"machineType"* name=*"machine"* maxOccurs=*"unbounded"* minOccurs=*"0"*>

</xs:element>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"orderType"*>

<xs:sequence>

<xs:element type=*"wishesType"* name=*"foods"*/>

<xs:element type=*"wishesType"* name=*"drinks"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"OrdersType"*>

<xs:sequence>

<xs:element type=*"orderType"* name=*"order"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"randomizationType"*>

<xs:simpleContent>

<xs:extension base=*"xs:string"*>

<xs:attribute type=*"xs:byte"* name=*"qtd"* use=*"optional"*/>

<xs:attribute type=*"xs:string"* name=*"set"* use=*"optional"*/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:complexType name=*"customersSNType"*>

<xs:sequence>

<xs:element type=*"randomizationType"* name=*"randomization"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"statementType"*>

<xs:sequence>

<xs:element type=*"blockType"* name=*"block"*/>

</xs:sequence>

<xs:attribute type=*"xs:string"* name=*"name"*/>

</xs:complexType>

<xs:complexType name=*"missionType"*>

<xs:sequence>

<xs:element type=*"sliderType"* name=*"slider"* maxOccurs=*"unbounded"* minOccurs=*"0"* />

<xs:element type=*"explanationType"* name=*"explanation"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"hintsType"* name=*"hints"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"selectMachineType"* name=*"selectMachine"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"commandsType"* name=*"commands"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"xs:string"* name=*"cooker"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"customersSNType"* name=*"customersSN"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"customersType"* name=*"customers"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"objectivesType"* name=*"objectives"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

<xs:element type=*"xmlType"* name=*"xml"* maxOccurs=*"unbounded"* minOccurs=*"0"*/>

</xs:sequence>

<xs:attribute type=*"xs:boolean"* name=*"open"* use=*"optional"*/>

<xs:attribute type=*"xs:integer"* name=*"timeLimit"* use=*"optional"*/>

</xs:complexType>

</xs:schema>

# Instructions for use

## User guide

The structure and organization of the user interface are described in XXX. But it is good to remember how the UI is organized:

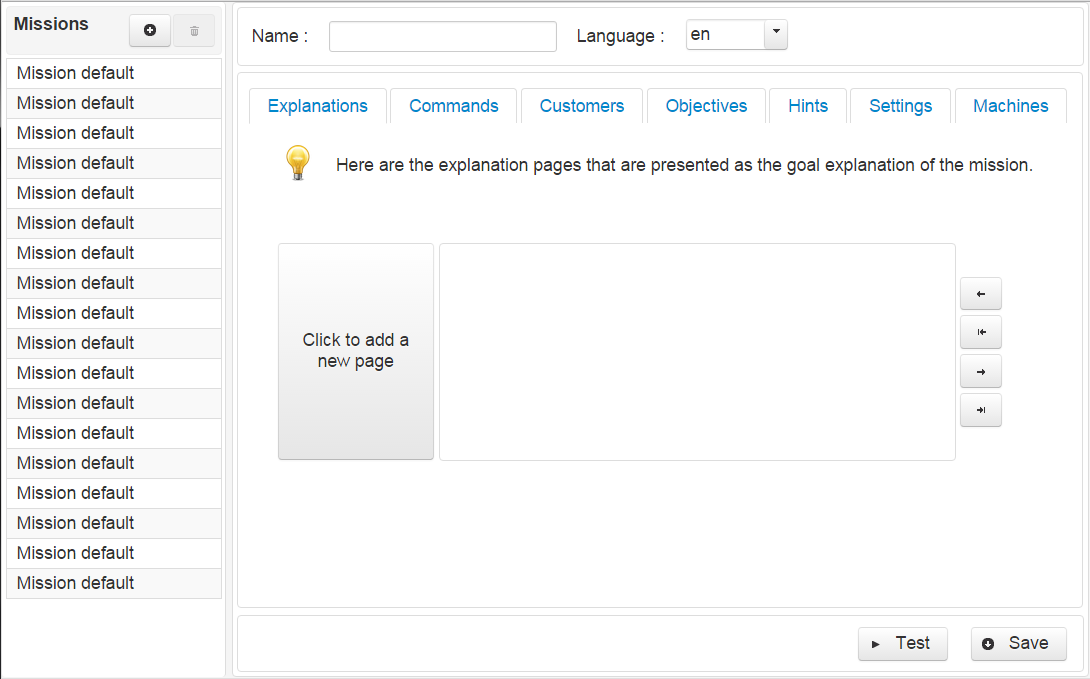


Figure 39 User interface

This is the first page which user will see. On the left side are the available missions in database. You can choose one to edit or remove. Also, you can create a new mission by choosing the button with icon ‘+’.

Let’s begin with creating a new mission. It does not matter the order of starting. You can start with any option you would like: explanations, customers and so on. If you miss or did something wrong you will be alerted by message.

* Explanations

In figure 39 is shown how ‘explanations’ looks like. First of all, you have to press the big button to add a new page and it will appear the dialog from figure 40.

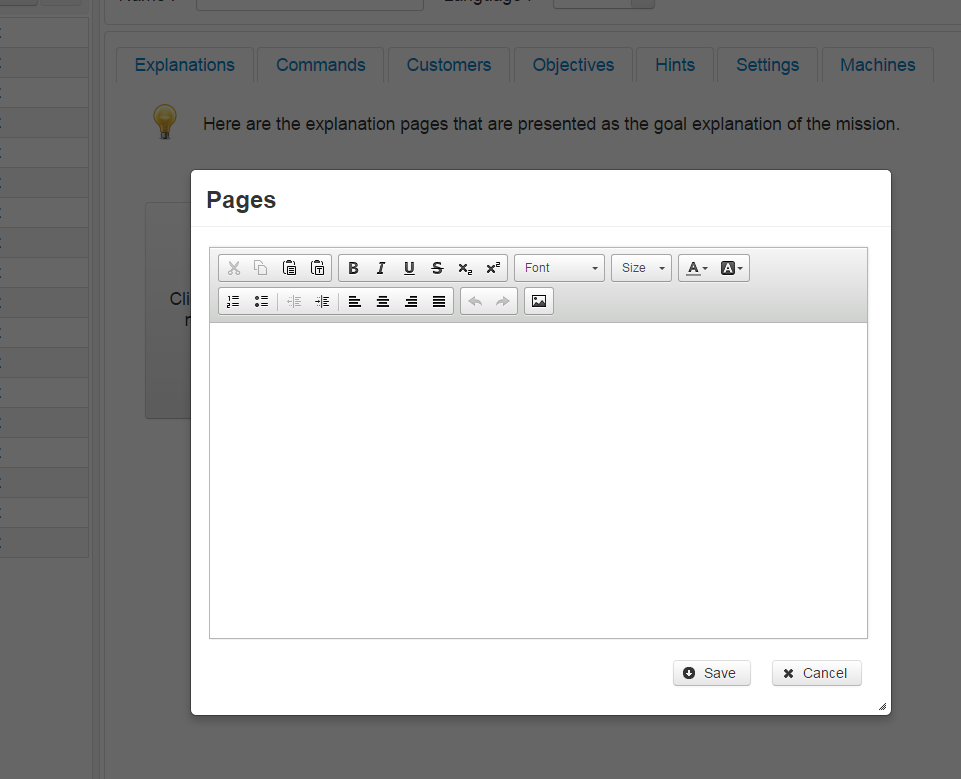


Figure 40 Add new page

You can enter formatted text and add images, as well (figure 41). To save the page and check if everything is okay you can press button ‘Save’ (figure 42) or ‘Cancel’ – to close the dialog.

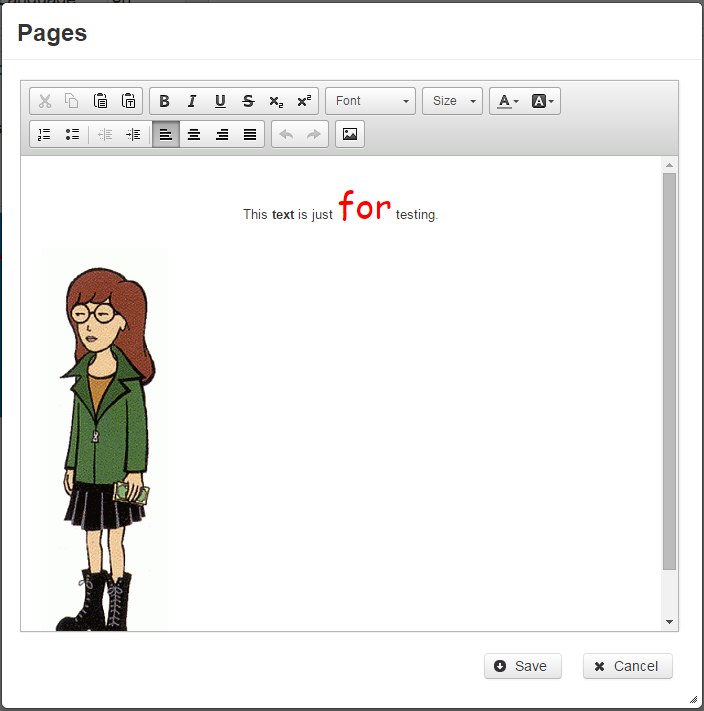


Figure 41 Explanation page - filled out with no errors

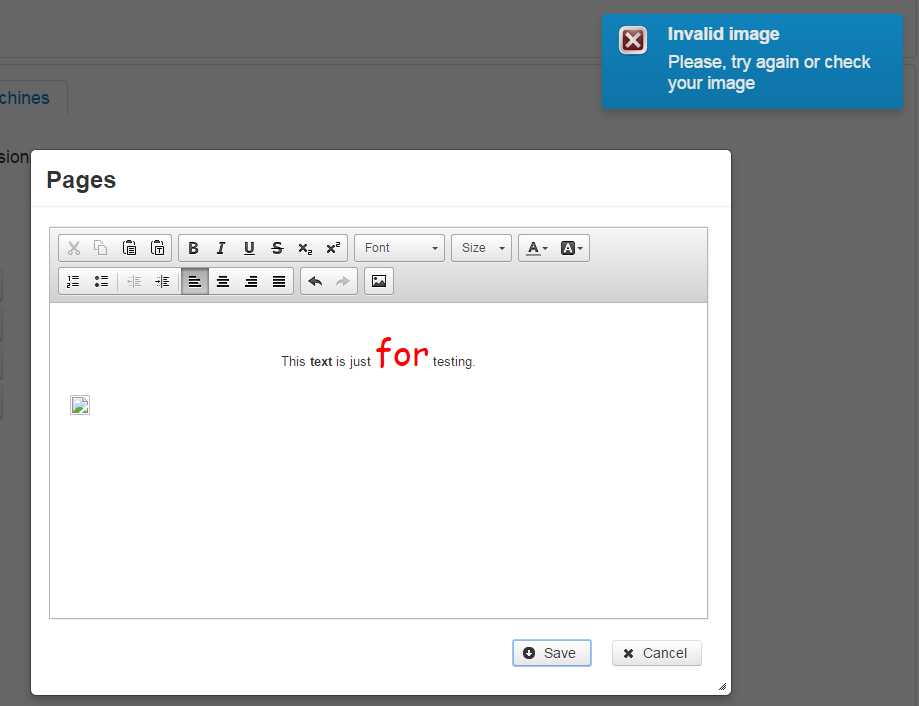


Figure 42 Explanation page - invalid image path or format

By pointing a page two buttons will appear, you can use them to delete or edit pages (figure 43). Also, there are four buttons with arrows for reordering the pages.

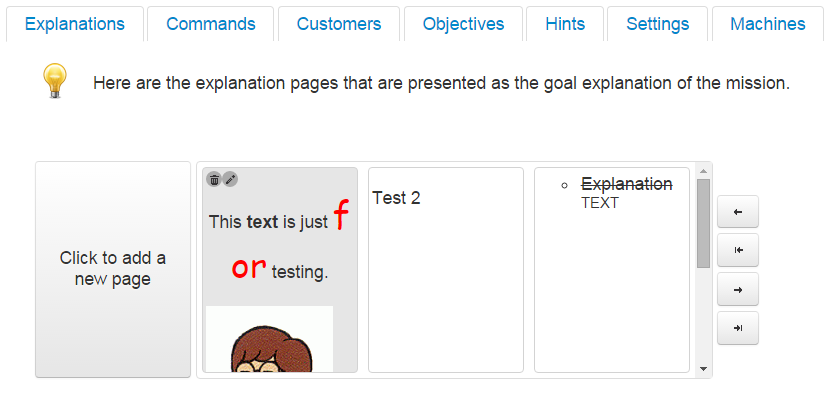


Figure 43 Editing/Removing pages

* Commands

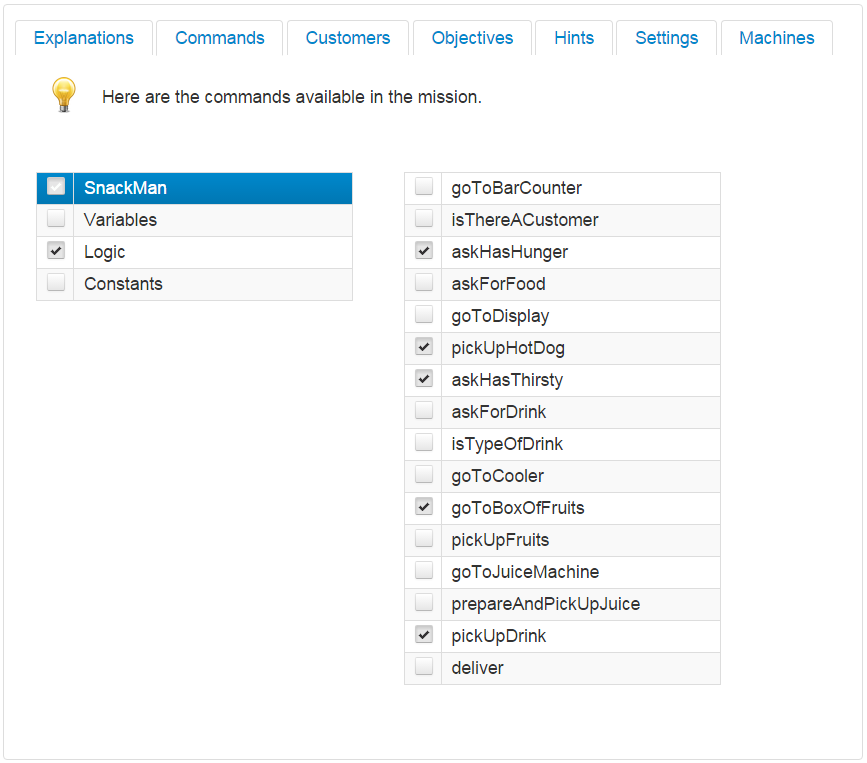


Figure 44 Commands

Here you can choose the commands you want to use in the game. Using the first list you can choose the ‘root’ commands. Once you select an option the ‘child’ commands will appear onto the right list and you can select or deselect commands by the check boxes.

* Customers

It is possible to add maximum 12 customers. Once you add a customer, its icon will be changed (figure 45):

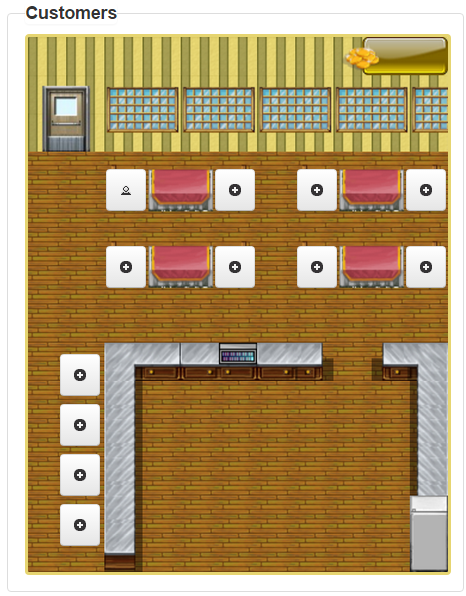


Figure 45 Customers in "Snack Bar"

If you click one of the buttons it will be shown a dialog with options for adding, editing and removing orders, foods and drinks. You can choose the skin of the customer, starting and destination positions, as well (figure 46).

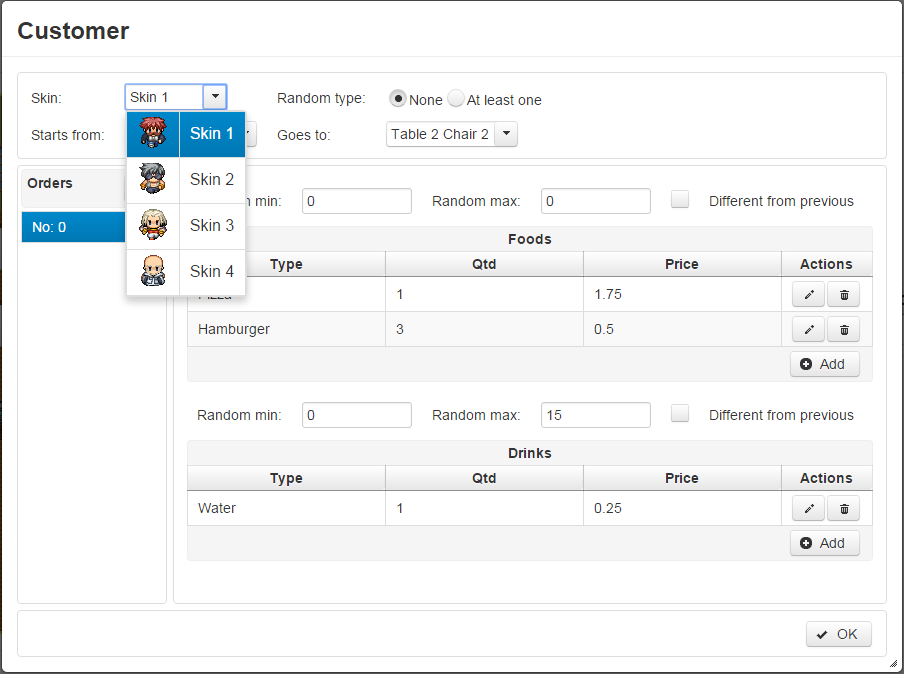


Figure 46 Customer dialog

* Condition Builder

The dialog for building a condition is shown in figure 47. To add a sub-condition you have to press button ‘Add’ and the ‘condition’ panel will be enabled. Firstly, you have to choose a function (figure 48).

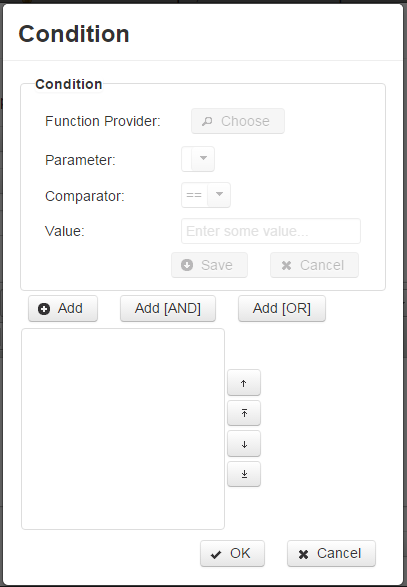


Figure 47 Condition dialog

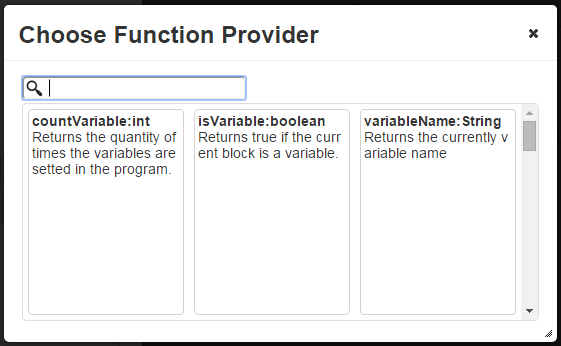


Figure 48 Function provider

There are a lot of functions. If you want to filter them by their name or type you can just enter some text in the search box.

Then you have to choose parameter (if it is available), comparator and enter some value and click button ‘Save’. If an error occurs you will be notified by message.

For adding logical operators such as ‘and - &&’, ‘or - ||’ you can use the buttons ‘Add [AND]’ and ‘Add [OR]’.

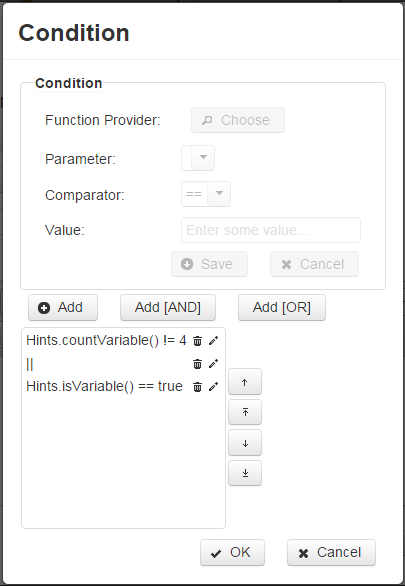


Figure 49 Condition list

Sub-conditions and logical operators can be edited or removed or reordered, similarly to explanation pages.

* Settings

Some additional settings can be made in this section: to set time limit, initial position of the cook and so on (figure 50).

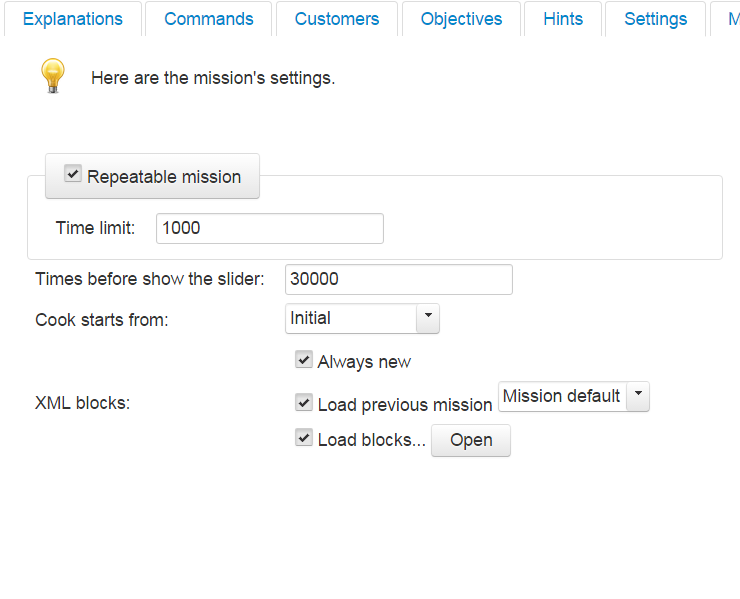


Figure 50 Settings

If you choose ‘Load blocks…’ and click on the button ‘Open’ you can build some block expressions (figure 51). They will be used in the game as starting point when students solve some problem.



Figure 51 Building block expressions

After these steps the last thing you have to do is to enter the name of the mission. Then you can save it (figure 52).

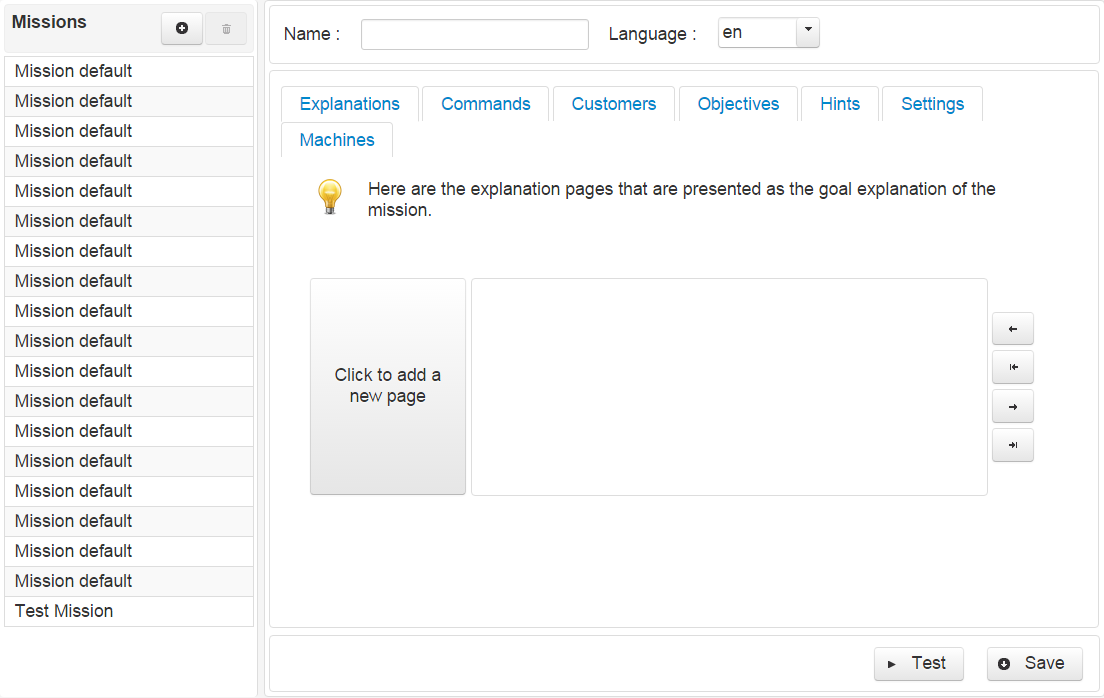


Figure 52 Created mission "Test Misison"

## Instructions and requirements for installation of the system

In order to run this app in your local machine you need:

* Java Development Kit (JDK)
* Eclipse IDE
* Web-server (for example Tomcat 7.0)
* The database (MySQL also)
* Web browser

The PrimeFaces libraries are included in the project but if they are not you need them too.

Here are the steps you have to do:

1. Install Java JDK and Eclipse IDE (Luna or Mars are recommended versions);
2. Install Tomcat v7.0 via Eclipse;
3. Import the project from Git;
4. Create a new server and choose Tomcat v7.0;
5. Add the project onto the server;
6. Add the ‘db.properties’ file in folder ‘project\_name/web/META-INF’;
7. Install the database – you need MySQL and you can use MySQL Workbench tool to manage your database;

## Instructions for maintenance of the system

Once the system is installed, the main thing you might worry about is to preserve the integrity of the database. Because of that it would be good the administrators of the server to make backups and archives of the database.

## Hardware requirements

Hardware requirements are minimal. The application is developed on laptop with processor Intel Core i7-2670QM CPU @ 2.20 GHz, 8 GB of RAM and 750 GB HDD. But this application can be run on computers with weaker configuration. Since the requirements for this type of applications are not too high.

# Test Results

The application was tested with different test scenarios in the process of developing. A lot of bugs were fixed, but the main project is still in process of developing (at the moment of writing this paper) and it might be necessary making further changes. The list of issues which occurred and were fixed can be seen on GitHub address of the project - <https://github.com/adilsonv77/nobugssnackbar/issues> (figure 53).

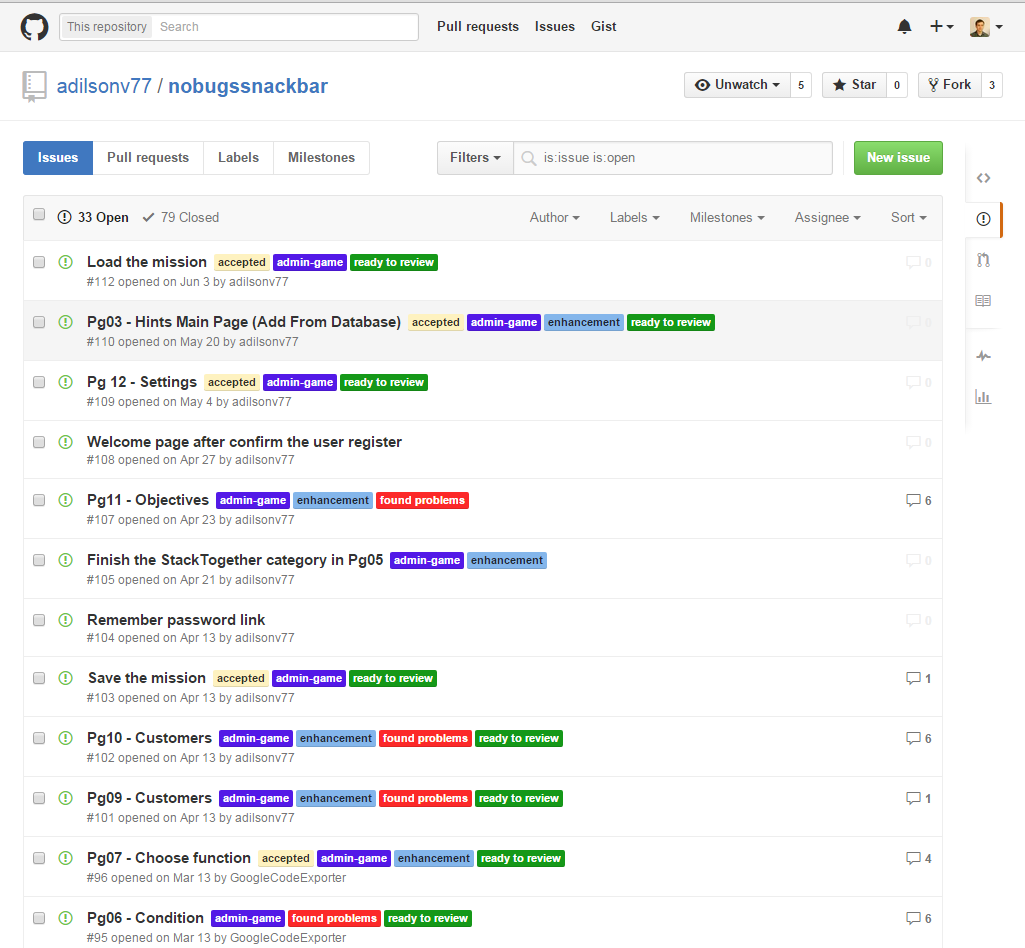


Figure 53 Issues - GitHub.com

Also, the application was tested on different browsers (Google Chrome, Mozilla Firefox, Internet Explorer, Opera and Microsoft Edge). It looks a little bit different only on Microsoft Edge, but it keeps its functionality.

# Main results

A software for creating and maintaining of missions in an introductory computer programming serious game engine, called “NoBug’s Snack Bar”, was designed and implemented. It allows to users to create their own missions and maintain the available missions in the game.

In process of developing occurred the following difficulties and issues:

* Creating of the XSD file

Firstly, the XSD file was created manually. But it is not the best way for doing this. So, a tool for automatically generating of XSD from XML was used for this purpose ([freeformatter.com](http://www.freeformatter.com/xsd-generator.html)) and additional treatment, as well.

* Understanding the concept of Model–View–Controller (MVC) architectural pattern
* PrimeFaces:
  + Treating of events
  + Sometimes it is necessary to create a convertor for some models, and sometimes it is not
  + Changing the UI of the PrimeFaces elements (*e.g. p:orderList in “Explanation”*)
  + Manipulation of the objects (*delete, edit, reorder, etc.*)
  + Tag nesting (*p:layout – p:dialog*)
* Marshalling of the business objects

The objects were marshalled using code which was not ‘flexible’, easily maintainable and understandable and later it turned out that there is a Java technology for this purpose, called Java Architecture for XML Binding (JAXB).

# Conclusions and recommendations

Some of the technologies and concepts used in the project were new for me and I suppose they are not used properly (in hundred percent). Thus, it is a good idea to review entire code, although the application accomplishes goals of the project.

# References

1. <https://en.wikipedia.org/wiki/Unified_Modeling_Language>, 13.09.2015

1. <https://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/UML_diagrams_overview.svg/808px-UML_diagrams_overview.svg.png>, 13.09.2015

1. <http://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp>, 13.09.2015

1. <https://en.wikipedia.org/wiki/Activity_diagram>, 13.09.2015

1. <https://www.techopedia.com/definition/27489/activity-diagram>, 13.09.2015

1. <http://www.w3schools.com/html/html_entities.asp>, 13.09.2015

1. <https://en.wikipedia.org/wiki/Package_diagram>, 13.09.2015

1. <https://en.wikipedia.org/wiki/Class_diagram>, 13.09.2015

1. <https://en.wikipedia.org/wiki/Sequence_diagram>, 13.09.2015

1. <https://en.wikipedia.org/wiki/Systems_architecture>, 13.09.2015

1. <https://en.wikipedia.org/wiki/Conceptual_schema>, 13.09.2015

1. <http://mobiledevices.about.com/od/additionalresources/fl/Comparing-Java-IDEs-Eclipse-vs-NetBeans-vs-IntelliJ.htm>, 13.09.2015

1. <https://en.wikipedia.org/wiki/Database>, 13.09.2015

1. <https://en.wikipedia.org/wiki/MySQL>, 13.09.2015

1. <http://www.oracle.com/us/products/mysql/overview/index.html>, 13.09.2015

1. <http://devrates.com/stats/index>, 13.09.2015

1. <http://www.primefaces.org/whyprimefaces>, 13.09.2015

1. <https://en.wikipedia.org/wiki/PrimeFaces>, 13.09.2015

1. <http://www.primetek.com.tr/products.html>, 13.09.2015

1. <https://en.wikipedia.org/wiki/GitHub>, 13.09.2015

1. <http://searchsoa.techtarget.com/definition/XSD>, 13.09.2015

# Attachments

A CD which includes the project with all needed applications and files, the UML diagrams and the current paper in e-format.

Currently (19.09.2015), the project is available on <https://github.com/adilsonv77/nobugssnackbar/> or <https://github.com/thesilent/nobugssnackbar> and the game can be tested on <http://nobugssnackbar.dei.uc.pt/> **but it is still developing**.