FINKInformator

TECHNICAL DOCUMENTATION

TABLE OF CONTENTS

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms and Abbreviations

2 System Overview and characteristics

2.1 System Architecture

2.2 Infrastructure Services

3 System Design

3.1 Documentation Standards

3.2 Naming conventions

3.3 Programming Standards

3.4 Software development tools

4 Component Description

**1 INTRODUCTION**

**FINKInformator is a web application created to inform FINKI students about optional courses that students have to choose from in their education as well as about college mandatory courses. Application displays course description and also collects comments from students themselves and displays the comments in items representing college courses. FINKInformator is an application developed to work on multiple types of devices such as computers, mobile phones and iPads. No authentication is required for accessing the application so its usage offers no security risks.**

* 1. **PURPOSE**

Purpose of this document is to describe and present FINKInformator web application to readers, possible users, developers or maintainers.

* 1. **SCOPE**

FINKInformator is a web application created mainly for FINKI students where they can inform about past experiences of other students who had intended college courses. Hopefully, this application will help students make the right choice when choosing from variety of optional courses and also get familiar with matter of interest of obligatory courses. It was developed using operating system Windows and programming language C# in Microsoft Visual Studio 2015 IDE.

* 1. **DEFINITIONS, ACRONYMS AND ABBREVIATIONS**

|  |  |
| --- | --- |
| *FINKI (FCSE)* | *Faculty of Computer Science and Engineering in Skopje, Macedonia* |
| *Class Diagram* | *Describes the structure of a system* |
| *Entity Relationship (ER) Diagram* | *Illustrates system's entities and the relationships between those entities* |
| *Integrated development environment (IDE)* | *Software application that provides comprehensive facilities to computer programmers for software development* |

* 1. **READING INSTRUCTIONS**

All information in this document is number organized in headings and subheadings. Other documentation and information referred to in this document will be correctly designated and pointed out and may be used as additional information to different kinds of readers.

1. **ARCHITECTURAL OVERVIEW**

FINKInformator is based on three-tier architecture. It is developed using Code First approach offered by Visual Studio Entity Framework version 6.1.3. Three-tier architecture was chosen to help developers separate tasks and concerns while working on the project.

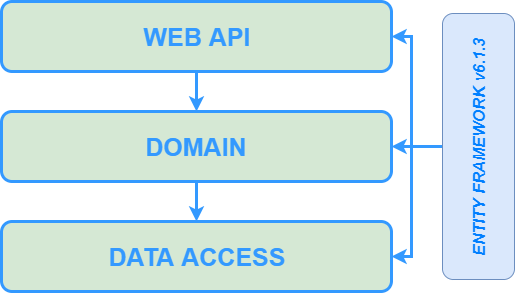
* 1. **SYSTEM ARCHITECTURE**

The system is based on three-tier client-server architecture consisting of presentation, logic and data tier. The presentation tier is developed using Angular2 web framework. Application’s web page collects basic information about which year and semester the user is interested in informing about and then offers information about all mandatory and optional available courses displayed as items on the screen. The web page was designed following the principles of responsive and simple design.

The logic and data layer were mainly covered using the Code First Entity Framework version 6.1.3 approach offered by Visual Studio Enterprise 2015.

Data layer is the layer responsible for accessing information from the database. FINKInformator uses SQL database.

* 1. **DOMAIN**



*Figure 1. Domain overview*

Domain design includes four classes. Class names are: Course, Program, ProgramsCourses and CoursesPrerequisites.

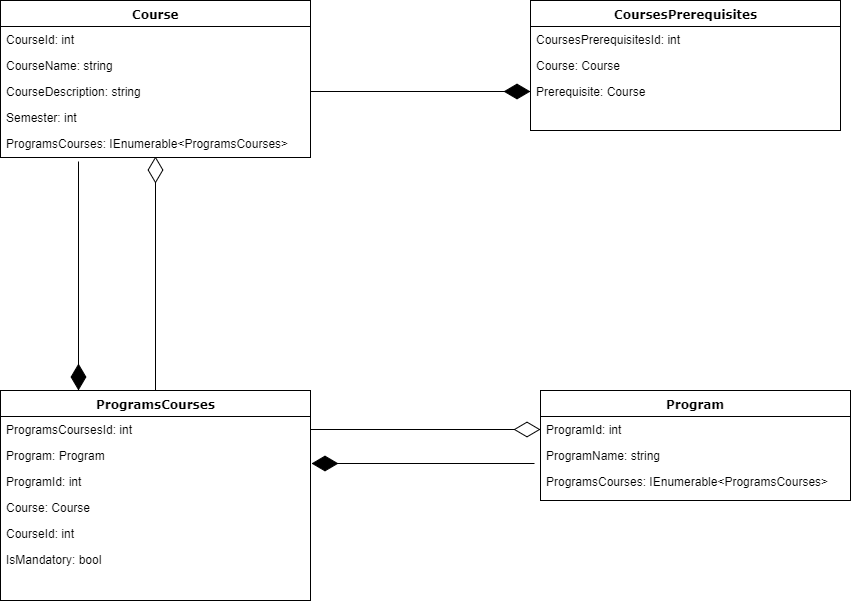
The Course class represents college course. This class contains CourseId – an integer needed for unique identification of the courses, CourseName – string representing the name of the course used at the university, CourseDescription – string limited to 2000 chars explaining matter of interest and content of the course, Semester – an integer varying from 1 to 8 to identify the semester in which the course is offered and ProgramsCourses – an IEnumerable of ProgramsCourses class objects needed for creation of relationships between the database tables by the Entity Framework version 6.1.3.

The Program class represents college program teaching multiple courses. This class contains ProgramId – an integer for unique identification of the college program and ProgramName – string representing the name of the program used at the university.

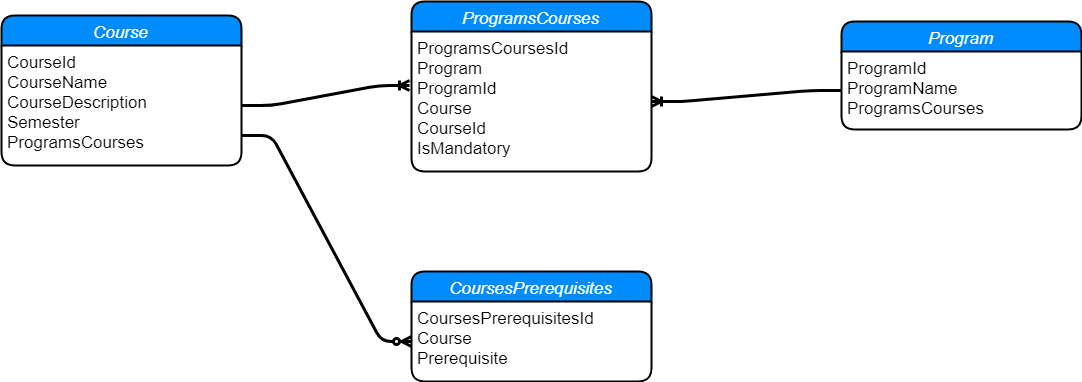
The ProgramsCourses class represents a pair of college course and a college program offering the specific course. ProgramsCourses class contains ProgramsCoursesId – an unique identificator of ProgramsCourses data entry, Program class object named Program and its ProgramId (an integer), Course class object and its CourseId and finally a bool property indicating if the course is mandatory in the specific program.

The CoursePrerequisites class also represents a pair of entities, but this time both of them college courses – a course and a course prerequisite for the first one. This class consist of CoursesPrerequisitesId – an integer, data entry identificator in the database table and two Course class objects named Couse and Prerequisite.

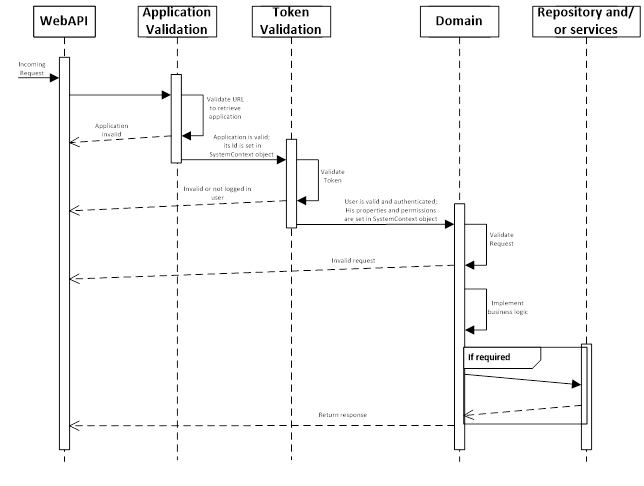
**3 DIAGRAMS**



*Figure 2. Class Diagram for the Business logic of the application*

**

*Figure 3. ER Diagram for the Business logic of the application*



*Figure 3. ER Diagram for the Business logic of the application*

1. **INFRASTRUCTURE SERVICES**

From infrastructure services there are two IIS projects: first one for the API and the second one for the UI of the application. FINKInformator uses SQL server.

**5 DOCUMENTATION STANDARDS**

Code complexity is stable and does not vary among different code areas. There might be exception with LINQ expressions which are slightly more complex to understand so this code area should be supported by more comments than the others.

**6 NAMING CONVENTIONS**

FINKInformator is consistent with all .NET naming conventions and standards listed below:

*1) Use camelCasing for parameters;*

*2) Use PascalCasing for class names, enumeration type, enumeration values, events, exception class names, read-only static field names, interfaces, methods, namespaces, resource keys and properties;*

*3) Consider ending the name of derived classes with the name of the base class;*

*4) Do prefix interface names with the letter I to indicate that the type is an interface;*

*5) Do ensure that when defining a class/interface pair where the class is a standard implementation of the interface, the names differ only by the letter I prefix on the interface name;*

*6) Do choose easily readable identifier names;*

*7) Do favor readability over brevity;*

*8) Do name generic type parameters with descriptive names, unless a single-letter name is completely self-explanatory and a descriptive name would not add value;*

*9) Consider using the letter T as the type parameter name for types with one single-letter type parameter;*

*10) Do prefix descriptive type parameter names with the letter T;*

*11) Consider indicating constraints placed on a type parameter in the name of parameter;*

*12) Do use a singular name for an enumeration, unless its values are bit fields;*

*13) Do add the suffix Attribute to custom attribute classes;*

*14) Do add the suffix EventHandler to names of types that are used in events;*

*15) Do add the suffix Callback to the name of a delegate that is not an event handler;*

*16) Do add the suffix Exception to types that inherit from System.Exception;*

*17) Do add the suffix Dictionary to types that implement System.Collections.IDictionary or System.Collections.Generic.IDictionary<TKey, TValue>;*

*18) Do add the suffix Collection to types that implement System.Collections.IEnumerable, System.Collections.ICollection, System.Collections.IList, System.Collections.Generic.IEnumerable<T>, System.Collections.Generic.ICollection<T>, or System.Collections.Generic.IList<T>;*

*19) Do add the suffix Stream to types that inherit from System.IO.Stream;*

*20) Do add the suffix Permission to types that inherit from System.Security.CodeAccessPermission or implement System.Security.Ipermission;*

*21) Do add the suffix EventArgs to classes that extend System.EventArgs;*

*22) Do use the dot separator (".") to nest identifiers with a clear hierarchy;*

*23) Do use semantically interesting names rather than language-specific keywords for type names;*

*24) Do use a generic common language runtime (CLR) type name, rather than a language-specific name, in the rare cases when an identifier has no semantic meaning beyond its type;*

*25) Do use a common name, such as value or item, rather than repeating the type name, in the rare cases when an identifier has no semantic meaning and the type of the parameter is not important;*

*26) Do prefix namespace names with a company name to prevent namespaces from different companies from having the same name and prefix;*

*27) Do use a stable, version-independent product name at the second level of a namespace name;*

*28) Do use Pascal casing, and separate namespace components with periods;*

*29) Consider using plural namespace names where appropriate;*

*30) Do choose names for your assembly DLLs that suggest large chunks of functionality such as System.Data. Assembly and DLL names do not have to correspond to namespace names but it is reasonable to follow the namespace name when naming assemblies;*

*31) Do not use the same name for a namespace and a type in that namespace;*

*32) Do not use organizational hierarchies as the basis for names in namespace hierarchies, because group names within corporations tend to be short-lived;*

*33) Do not add the suffix Delegate to a delegate;*

*34) Do not use underscores, hyphens, or any other nonalphanumeric characters except for naming resources which can be named using underscores too;*

*35) Do not use Hungarian notation;*

*36) Avoid using identifiers that conflict with keywords of widely used programming languages;*

*37) Do not use abbreviations or contractions as parts of identifier names;*

*38) Do not use any acronyms that are not widely accepted, and then only when necessary;*

*39) Do not derive from the System.Enum class; use the keyword supported by your language instead;*

*40) Do not use prefix with enumerations and field names.*

**7 PROGRAMMING STANDARDS**

Programming standards followed by the team of developers of FINKInformator were:

1) Well-organized code supported by comments;

2) Modularity;

3) Simplicity.

Among these coding practices FINKInformator code as well satisfies .NET naming conventions.

**8 SOFTWARE DEVELOPMENT TOOLS**

Software development tools used while developing FINKInformator application were:

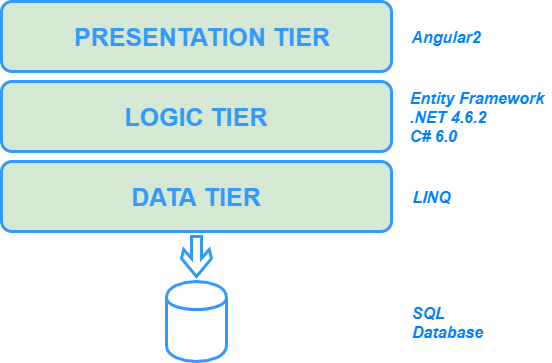
1. Microsoft Visual Studio Enterprise Edition 2015;

2) Angular web framework;

3) Microsoft Word;

4) Draw.io (Free online software for making flowcharts, process diagrams, org charts, UML, ER and network diagrams).

1. **COMPONENT DESCRIPTION**



*Picture 3. System components and architecture illustrated*