

FINKInformator

TECHNICAL DOCUMENTATION

*August, 2017*

TABLE OF CONTENTS

1 Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms and Abbreviations

1.4 Reading instructions

2 ARCHITECTURAL OVERVIEW

2.1 System Architecture

2.2 Models

3 diagrams

3.1 Class Diagram

3.2 Entity Relationship Diagram

3.3 Request Lifecycle Diagram

4 ENVIRONMENTS

4.1 Production Environment

5 INFRASTRUCTURE AND DISTRIBUTION

5.1 Infrastructure Services

5.2 Distribution

6 STANDARDS AND CONVENTIONS

6.1 Documentation Standards

6.2 Naming Conventions

6.3 Programming Standards

7 SOFTWARE DEVELOPMENT TOOLS

8 COMPONENT DESCRIPTION

|  |  |
| --- | --- |
| *Document name:* | FINKInformator Technical Documentation |
| *Version number:* | **04/09/2017** |
| *Date of version:* | **0.2** |
| *Applicable:* | **itLabs** |
| *Document owner:* | **Aleksandra Bozhinoska, Danilo Popovikj, Jovica Ilievski** |
| *Created by:* | **Aleksandra Bozhinoska** |
| *Approved by:* | **Kostadin Kadiev** |

**DOCUMENT CHANGE HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| Date: | Version: | Created by: | Change description: |
| 14/08/2017 | 0.1 | Aleksandra Bozhinoska | Initial version of the document |
| 04/09/2017 | 0.2 | Aleksandra Bozhinoska | Version updated corresponding to the change of technology used to develop the UI project |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**1 INTRODUCTION**

**FINKInformator is a web application created to inform FINKI students about optional and mandatory courses offered by the faculty. Application collects and displays comments and course description in items representing faculty courses. FINKInformator is an application developed to be used 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 OF THIS DOCUMENT**

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 faculty courses earlier. 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 mandatory courses. It was developed using operating system Windows and programming language C# in Microsoft Visual Studio 2015 IDE.

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

|  |  |
| --- | --- |
| *TERM* | *EXPLANATION* |
| *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. Definitions, acronyms and abbreviations that are used are explained in the previous subheading.

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, domain and data tier.

The presentation tier consists of two API controllers CoursesController and ProgramsController containing API Get, Put, Post and Delete methods.

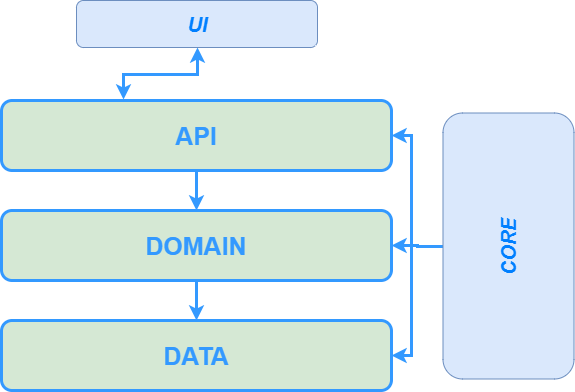
The domain tier consists of two types of classes: Managers and Validators. The managers (CoursesManager, ProgramsManager) manage the request/response logic of the data communication and the Validators (we used multiple validators for every type of request) validate the requests. For the needs of validation we used a third-party validation library – Fluent validation.

The data layer is the tier responsible for accessing information from the database. It was mainly covered by the Code First Entity Framework (version 6.1.3) approach offered by Visual Studio Enterprise 2015. We used two classes CoursesRespository and ProgramsRepository which contain the methods for data transfer implemented using Entity Framework Data Context (SchoolContext class placed in the same layer). FINKInformator uses SQL database.

The core of API FINKInformator project consists of classes representing models, requests, responses and interfaces.

The UI was implemented using Angular 2 platform. Application’s UI collects basic information about the program of studies, year and semester the visitor is interested in informing about and then displays information about mandatory and optional adequate courses. The web site was designed following the principles of responsive and simple design. FINKInformator’s UI also uses third-party platform for the functionality of collecting and displaying visitors’ comments and integration with social networks - Disqus.

API’s behavior is tracked using Nlog logger and can be tested using Swagger framework.



*Figure 1. Domain overview*

* 1. **MODELS**

Model design of FINKInformator includes four classes. Class names are: *Course*, *Program*, *ProgramsCourses* and *CoursesPrerequisites* (corresponding to the components of the faculty-programs-courses domain covered by the application).

The *Course class* represents faculty 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.

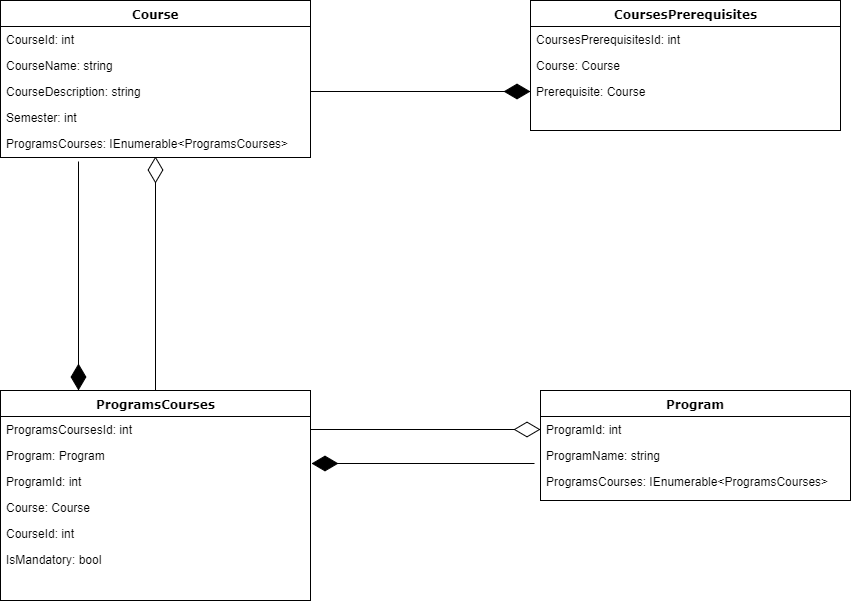
The *Program class* represents faculty program teaching multiple courses. This class contains ProgramId – an integer for unique identification of the faculty program and ProgramName – string representing the name of the program used at the university.

The *ProgramsCourses* class represents a pair of faculty course and a program offering that 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 boolean property indicating if the course is mandatory in the specific program.

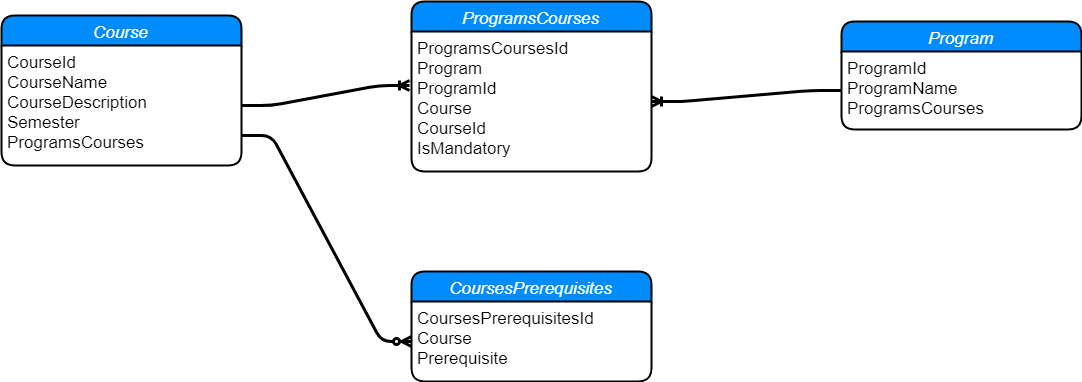
The *CoursePrerequisites* class also represents a pair of entities, but this time both of them faculty courses – a course and its prerequisite. This class consist of CoursesPrerequisitesId – an integer, data entry identificator in the database table and two Course class objects named Couse and Prerequisite.

1. **DIAGRAMS**

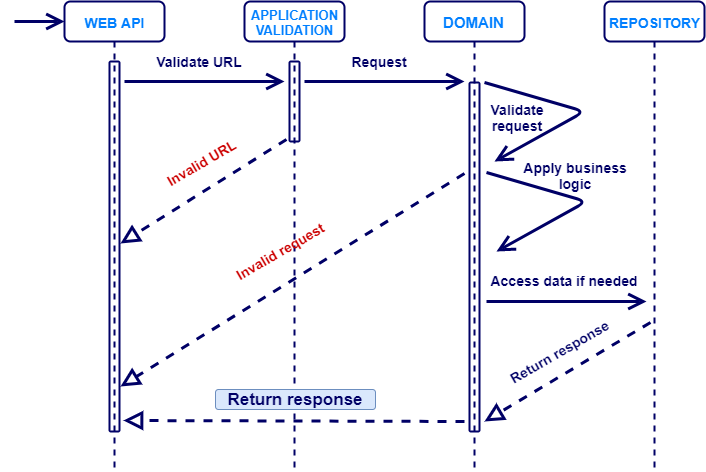
**3.1 Class Diagram**



**3.2 Entity Relationship Diagram**

**

* 1. **Request Lifecycle Diagram**



**4 ENVIRONMENTS**

**4.1 PRODUCTION ENVIRONMENT**

FINKInformator is on-premises software deployed on itLabs local IIS web server.

**5 INFRASTRUCTURE AND DISTRIBUTION**

**5.1 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.

**5.2 DISTRIBUTION**

Although FINKInformator is deployed on itLabs local server, its source code can be found on Github on this link: <https://github.com/itlabsinterns/FINKInformator>

Interested users can checkout and use the project as well as modify it.

**6 STANDARDS AND CONVENTIONS**

**6.1 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.

**6.2 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.*

**6.3 PROGRAMMING STANDARDS**

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

1) Well-organized code supported by comments;

2) Modularity of components (Dependency injection implemented);

3) Simplicity.

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

**7 SOFTWARE DEVELOPMENT TOOLS**

Software development tools used while developing FINKInformator application were:

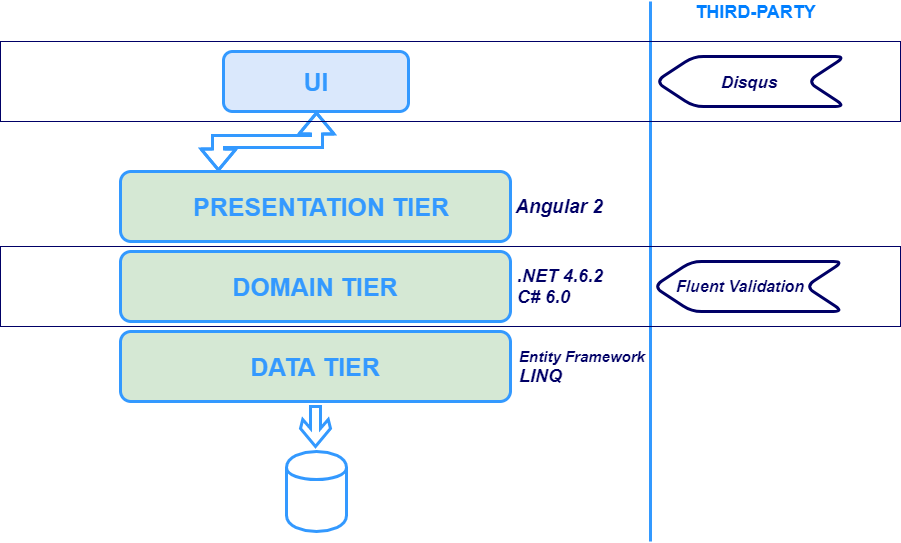
1. Microsoft Visual Studio Enterprise Edition 2015;
2. Microsoft Visual Studio Code;

2) Angular 2;

3) Microsoft Word;

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

1. **COMPONENT DESCRIPTION**



*Figure 2. System components and architecture overview*