# Design Document

This documentation tries to explain the contents of the application from a technical perspective and aims to be used as a design guide of the current functionality throughout the development phase and future improvements to the application as well as to act as a guide to any person that want to know the technical specifications of the application.

Any future changes should be translated into a new entry in the following table to keep track of the versions over the documentation and a summary of the changes done.

|  |  |  |
| --- | --- | --- |
| **Version** | **Updated by / date updated** | **Comments / changes** |
| 1.1 | Marcos Olivera - 10/13/2016 | Document created, versions table added and contents started to be added. |
|  |  |  |

## 1 – List of technologies and design patterns used

### Technologies used

* IDE: Microsoft Visual Studio Community 2015 with updates
* .NET Framework: Version 4.6.1
* Language (backend and service): C# 6.0
* ASP.NET MVC
* ASP.NET Web Api
* NuGet package manager to install required packages into Visual Studio 2015
* Mongo DB Server: ***Installed locally*** (version MongoDB 3.2.9 2008R2Plus Enterprise (64 bit)
* MongoDb-CSharp driver: ***Installed through NuGet Package Manager***
* C# Ninject (***All Installed through NuGet Package Manager(for detailed installation instructions please look at the README.md file***):
  + ***Ninject.Web.WebAp***
  + ***Ninject***
  + ***Ninject.Web.Common***
  + ***Ninject.Web.Common.WebHost***
  + ***WebActivatorEx***
* Unit test project: Created with the solution and updated accordingly

### Design patterns used

* MVC (Model View Controller) - Architectural pattern
* Dependency injection - Design pattern that implements inversion of control (IoC) design principle.

## 2 – Explanation of the architecture/design implementation and reasons of various choices

The application makes use of the following technologies and tools:

Visual studio community 2015 as the main IDE and C# as the main programming language, it uses the .NET Framework 4.6.1.

MVC architectural pattern is used, it provides a way of separation of concerns and gives the opportunity to write unit tests and automate manual testing; it is divided into 3 sections:

* Model - Represents the real world object and provides data to the view.
* View - Responsible for the look and feel.
* Controller - Responsible for taking the end user request and loading the appropriate Model and View.

Unit Test Project is integrated in the solution under the name ***UniversityLibraryWeb.Tests***, it contains the tests that will be executed in order to validate the correctness of the functionality.

ASP.NET Web Api used to expose data over HTTP following REST (Representational State Transfer) principles to be consumed by other clients.

NuGet package manager is used in the application to install and keep track of some packages that the application uses.

C# Ninject, it allows us to implement dependency injection into our application, the WebApi controllers include the injections for the services that execute the various operations required by the application.

Regarding the database we use MongoDB .bat/.json files so with it we can bulk load the data needed for the application (we need to query this data in order to support book searches), how the Publishers add the data about their books is out of scope.

Regarding the connection and interaction with the database, normally, we would have a separate project (*a.k.a* Data Layer) that handle the data interchange between the application and the database, in this application for the sake of simplicity the connectivity and exchange of information from/to the database is done directly in the UniversityLibraryWeb project.

### Solution structure

The solution named ***UniversityLibrary*** contains the following projects:

* UniversityLibraryWeb:
  + Is the main web application, it uses MVC and it is used to perform the required functionality that allows University Library Members to register and login to maintain their list of book demands, search for books from the publishers and place demands.
* UniversityLibraryWeb.Tests:
  + Is the test project for the solution, it contains all the tests we want to cover for the required functionality.

## 3 – Diagrams for component interaction, activity, sequence and/or classes of the important components.

Structure diagrams of application/components:

* Components diagrams **(to be added)**
* Classes diagrams:
  + ClassDiagram.cd has the uml diagram for classes, interfaces, etc.
  + ClassDiagram.png is just an export of the uml diagram in order to have a simple image to show the diagram.

Behavior diagrams of application/components:

* Activity diagrams **(to be added)**

Interaction diagrams of application/components:

* Sequence diagrams **(to be added)**