Plug

Learning Management System

Design Document

Contents

Document Outline

Document Purpose

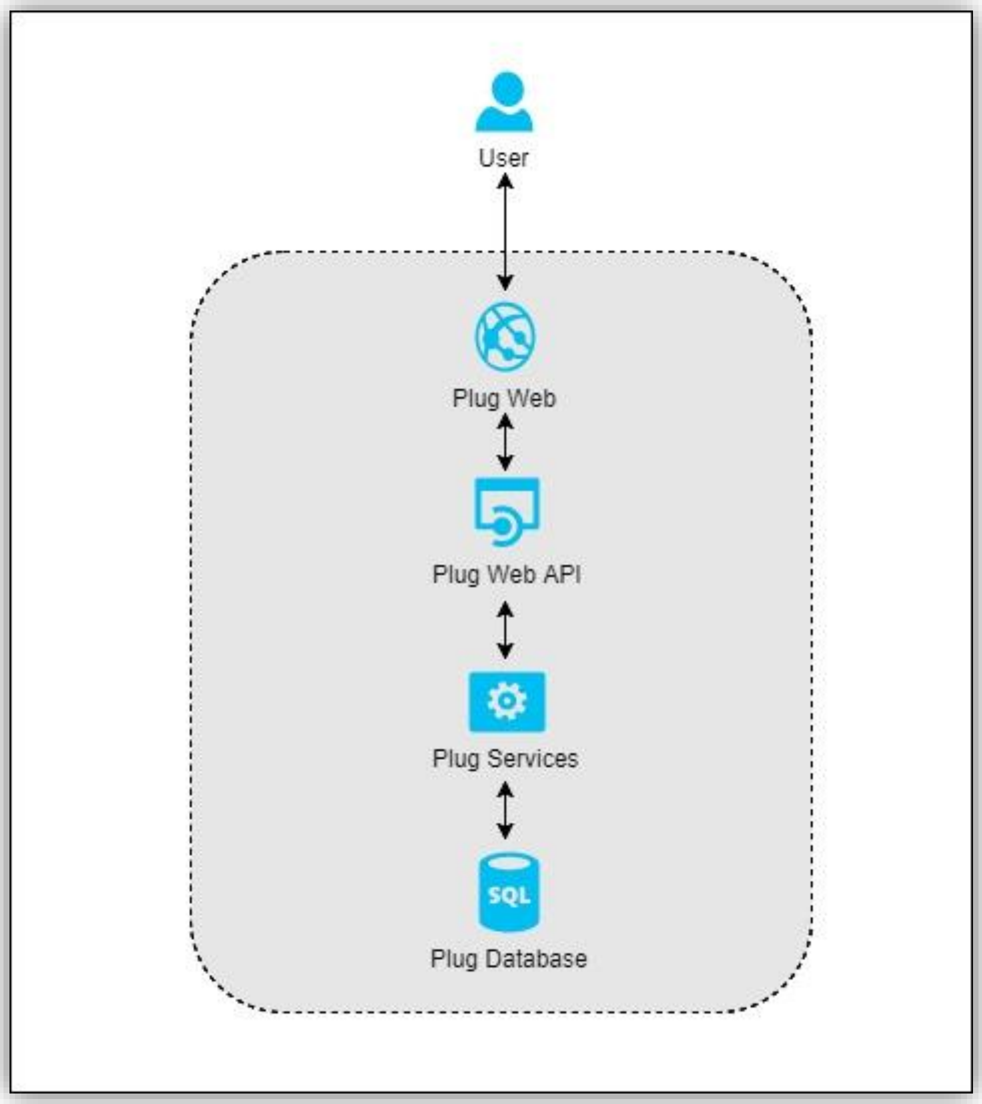
This document provides the high-level solution architecture design for the Plug Learning Management System (LMS). This document is the guideline for implementation across application development and infrastructure configuration. The detail design of infrastructure design, application design, and data dictionary are not part of content of this document.

Project Outline

1. Should be able to create and manage subject areas and modules under those subject areas.
2. Students / Learners should be able to subscribe to one or many subject areas.
3. Each module can consist different learning mechanisms such as a descriptive text, multimedia, multiple choice questions, etc.
4. Students / Learners can subscribe to any subject and they can complete each module depending on the interest.
5. During the process, you need to display the completion percentage of each subject area along with the last attended date for the LMS learnings.
6. When a user left a module in the middle of the progression, then later they should be able to continue it from the point they have stopped.
7. Any other features that you wish to include in the system.

Solution Overview

Solution Components



There are four major components of the Plug solution.

- Plug Web
- Plug Web API
- Plug Services
- Plug Database

The Plug web application will serve as the entry point of the Plug solution, users will access the web application to view and update information on the Plug solution.

Design and Implementation Principles
The solution design and construction will follow formal industry standards and best practices. The solution will be designed and developed using an object-oriented methodology and layered architecture. The solution will use the most suitable design patterns. The solution design will follow the .NET development guidelines. The solution will be available ONLY on English language.

Solution Technologies
.Net Core Framework, is the modular and high-performance implementation of .NET for creating web applications and services that run on Windows, Linux and Mac. It is open source and it can share the same code with .NET Framework and Xamarin apps.

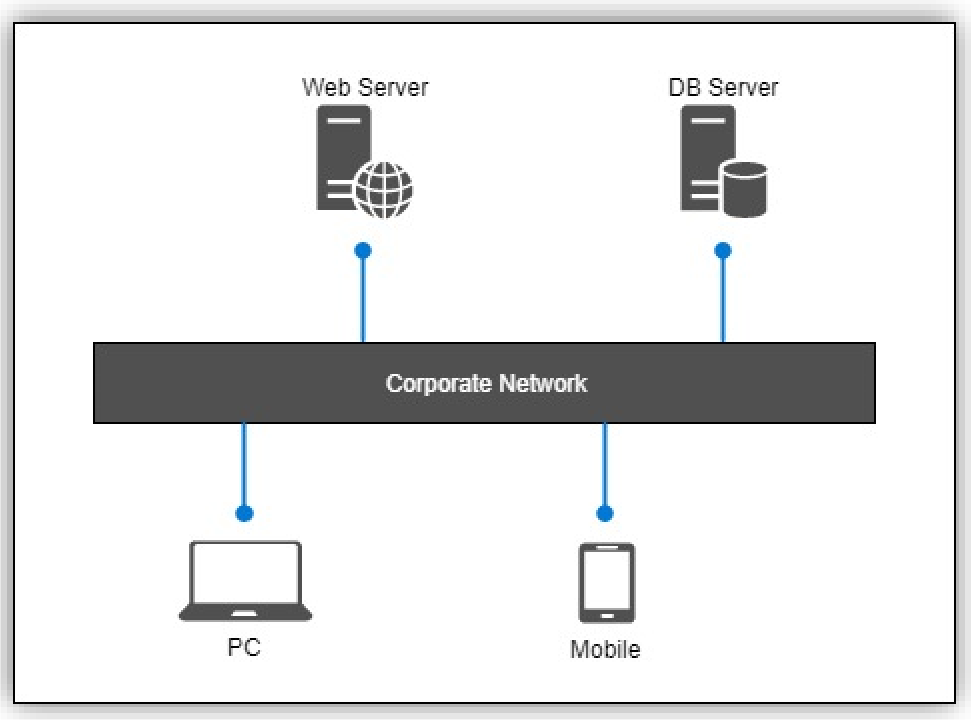
SQL Server, provides the underlying database services for applications and is, therefore, a fundamental underlying part of the solution.

Visual Studio 2017 Update 3, which provides a rich and productive application lifecycle management platform for the implementation team.

Development Technologies
The solution relies on certain technologies and several existing Microsoft Windows Server System products. The technologies, methodologies and development products involved in this solution include:

- .NET Core Framework 4.5
- C#
- Entity Framework Core
- ASP.NET Core Web API 2.0
- Layered Architecture
- Object Oriented Design and Programming
- Visual Studio
- HTML/CSS/JavaScript/Ajax
- Angular 4

Deployment View



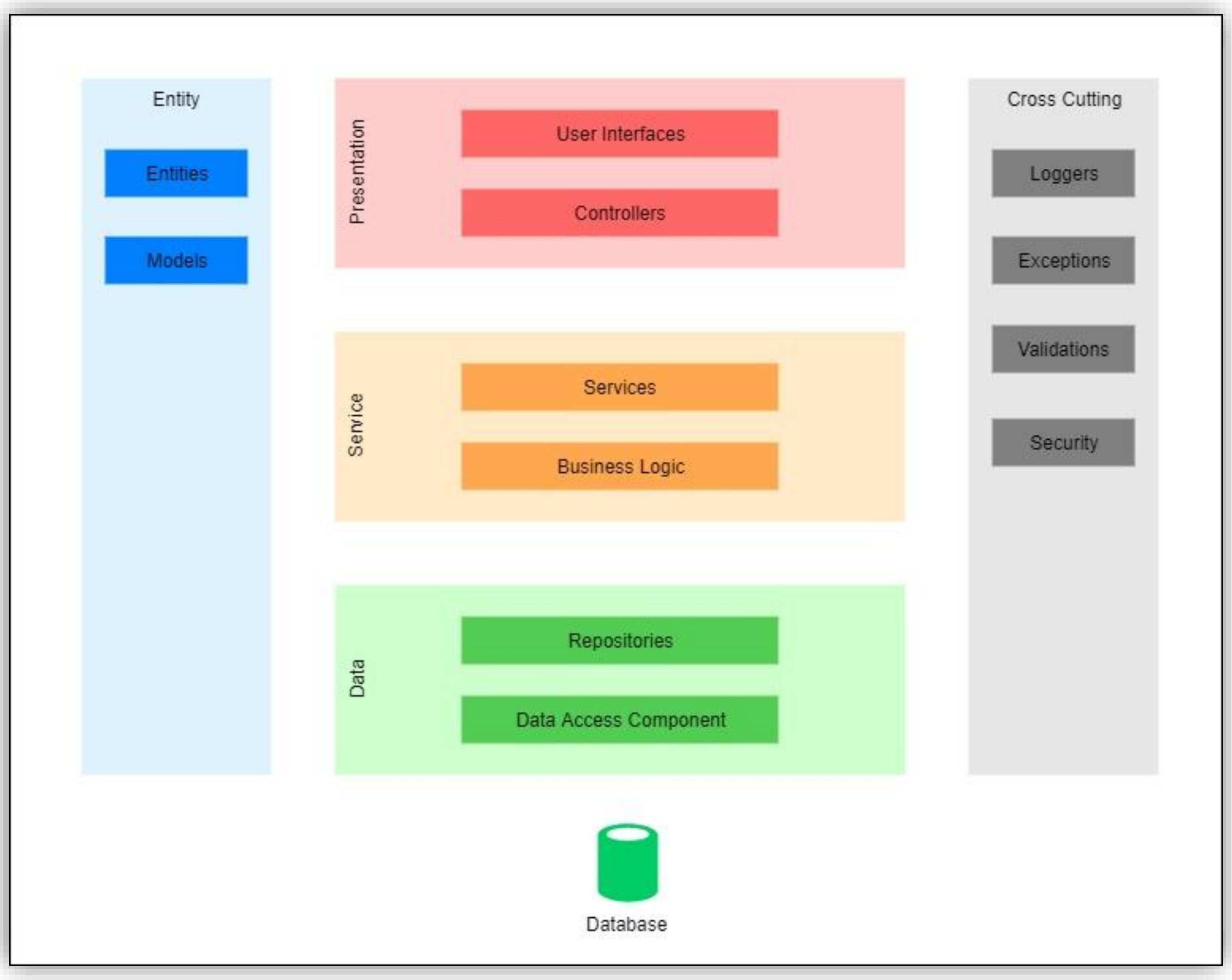
Web Server

This server giving the hosting facility to both Web Application and the Web API application.

DB Server

Hosts the database

Development View



Overall

The overall solution is divided into three layers: Presentation, Service and Data. The primary web site and web API will reside on Presentation layer. Data access components and Repositories will reside inside of data tier. Services and Business logics will reside inside of service layers.

Presentation Layer

Presentation layer code is the application's user interface that handles all interaction between the user and data. This is primarily implemented by using Angular 4 framework. To enhance usability and data integrity, common user interface elements will be developed by using components. A light weight HTTP based service layer (ASP .Net Core Web API) will be implemented as part of the web application to communicate with the clients.

Data Transfer Objects (DTOs)

Since the Services are implemented as a separate layer and exposing them as a ASP .Net Core application program interface, there can be multiple service calls required to satisfy the requirements of a single client request on the presentation tier. This could increase the overall response time beyond acceptable levels. As a solution, I will be using separate Data Transfer Object (DTO) to hold all the required data required for a request. As a result, this can simplify and optimize the service call signatures to accept a single DTO and to return a DTO.

Services

The services contain almost all of business logics defined within application, it interacts between presentation layer and data access layer by using DTOs and database entities as vessels, then applies all required business logic (such as validation, transformation, etc.).

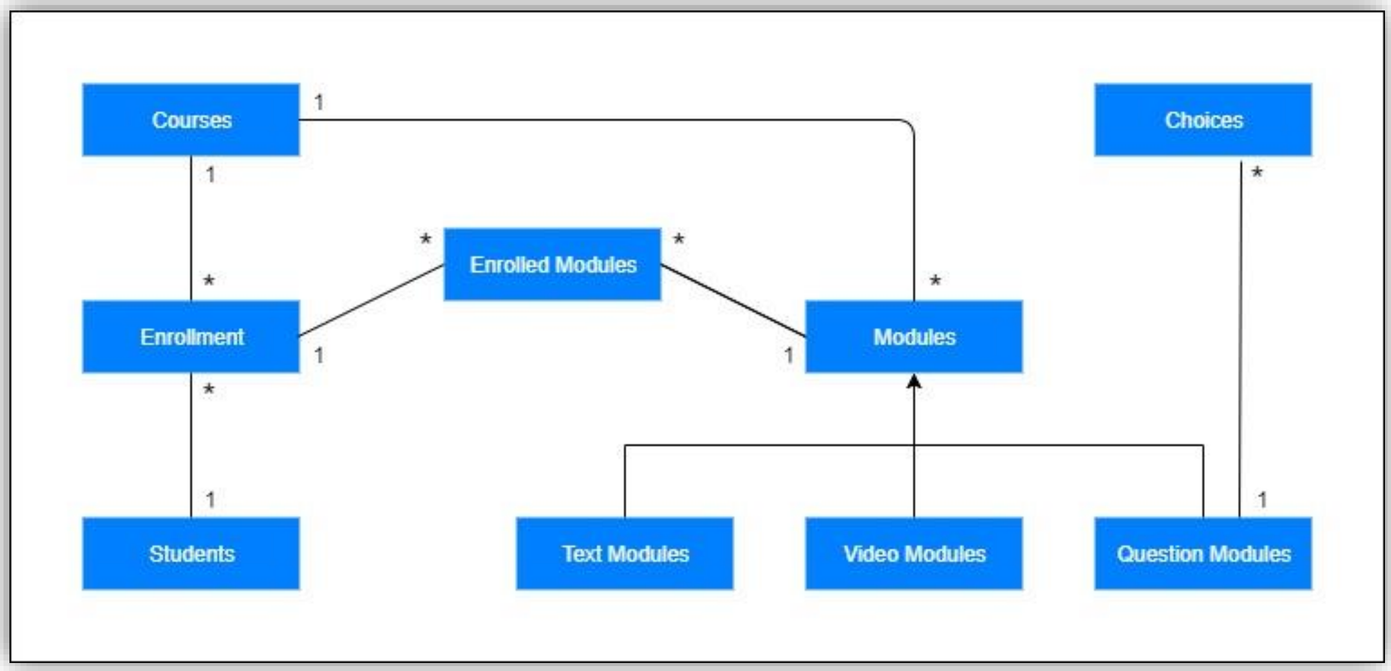
Data Access Layer

The data access layer provider access to database, it has the knowledge of underline data storage and all other details to retrieve or persist data. Primarily, the data access will be implemented using Microsoft Entity Framework Core. In addition, a unit of work and repository pattern based abstraction layer will be implemented to help insulate the application from changes in the data store and to facilitate automated unit testing or test-driven development (TDD). In addition, this abstraction layer will be responsible to implement transaction support (i.e. unit of work with single DB transaction).

Data Architecture

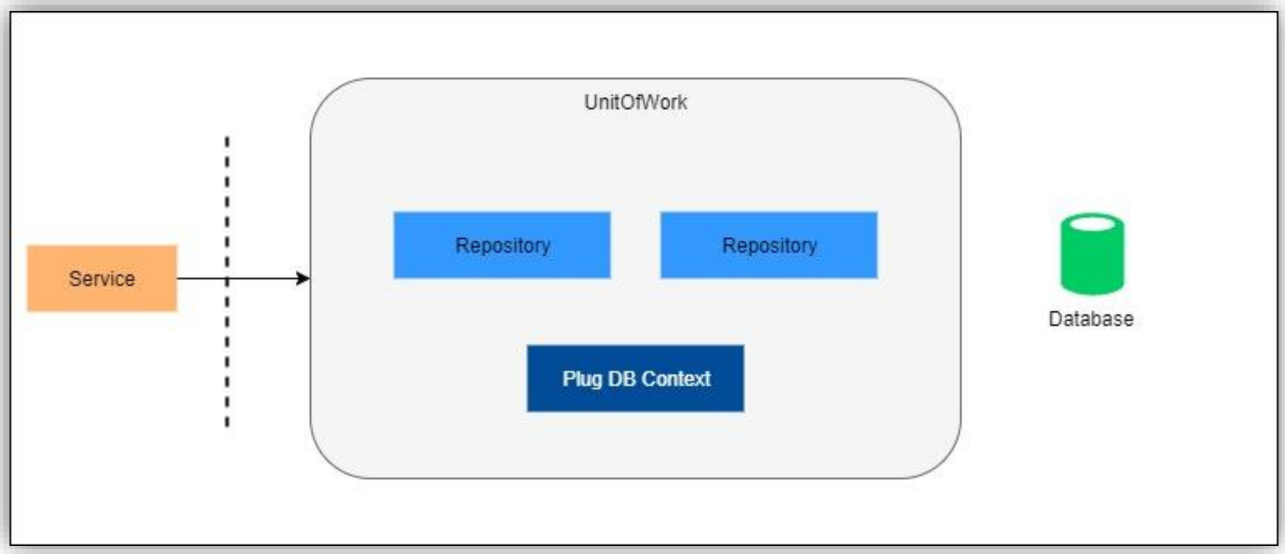
Business Entities

The following diagram shows the business entities related to the Plug Application



UnitOfWork

A unit of work and repository pattern based abstraction layer will be implemented to help insulate the application from changes in the data store and to facilitate automated unit testing or test-driven development (TDD). In addition, this abstraction layer will be responsible to implement transaction support (i.e. unit of work with single DB transaction).



Design Patterns

Singleton

Ensure a class has only one instance and provide a global point of access to it. In Plug application singleton design pattern has used to create a global interface for the DB Factory Class.

Factory

DB Factory provide an interface for creating families of related or dependent objects without specifying their concrete classes. At the movement Plug does not have multiple concrete class.

Repository

It is not a good idea to access the database logic directly in the business logic. Tight coupling of the database logic in the business logic make applications tough to test and extend further. Direct access of the data in the business logic may cause problems such as:

- Difficulty completing Unit Test of the business logic
- Business logic cannot be tested without the dependencies of external systems like database
- Duplicate data access code throughout the business layer (Don’t Repeat Yourself in code)

Dependency Injection

Dependency injection (DI) is a technique for achieving loose coupling between objects and their collaborators, or dependencies. Rather than directly instantiating collaborators, or using static references, the objects a class needs in order to perform its actions are provided to the class in some fashion.