Ciserro API

High Level Architecture &Design Document

|  |  |
| --- | --- |
| **Version Nos** | *1.0* |
| **Date** | *3-Aug-2016* |
| **Author** | *Ashin S* |
| **Approved by** |  |
| **Client Name** | *Ciserro* |

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Version #** | **Nature of change** | **Affected pages** | **Approved By** |
| *1* | *1.0* | *First Draft* | *All* |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[Introduction 5](#_Toc458060967)

[Goals 5](#_Toc458060968)

[Architecture Overview 5](#_Toc458060969)

[Layering Strategy 5](#_Toc458060970)

[Cross Cutting Concerns 6](#_Toc458060971)

[Authentication/Authorization 6](#_Toc458060972)

[Ciserro OAuth Server 6](#_Toc458060973)

[Communication 8](#_Toc458060974)

[Cryptography 8](#_Toc458060975)

[Configuration Management 8](#_Toc458060976)

[Caching 8](#_Toc458060977)

[Exception Management 8](#_Toc458060978)

[Auditing and Logging 9](#_Toc458060979)

[State Management 9](#_Toc458060980)

[Validation 9](#_Toc458060981)

[Globalization and Localization 9](#_Toc458060982)

[API Layer 9](#_Toc458060983)

[Business Logic Layer 9](#_Toc458060984)

[Data Sync Engine 9](#_Toc458060985)

[Payment Processor 9](#_Toc458060986)

[Data Layer 9](#_Toc458060987)

[Entity Framework 9](#_Toc458060988)

[Repositories 9](#_Toc458060989)

[Unit of Work 10](#_Toc458060990)

[Database 10](#_Toc458060991)

[References 10](#_Toc458060992)

# Introduction

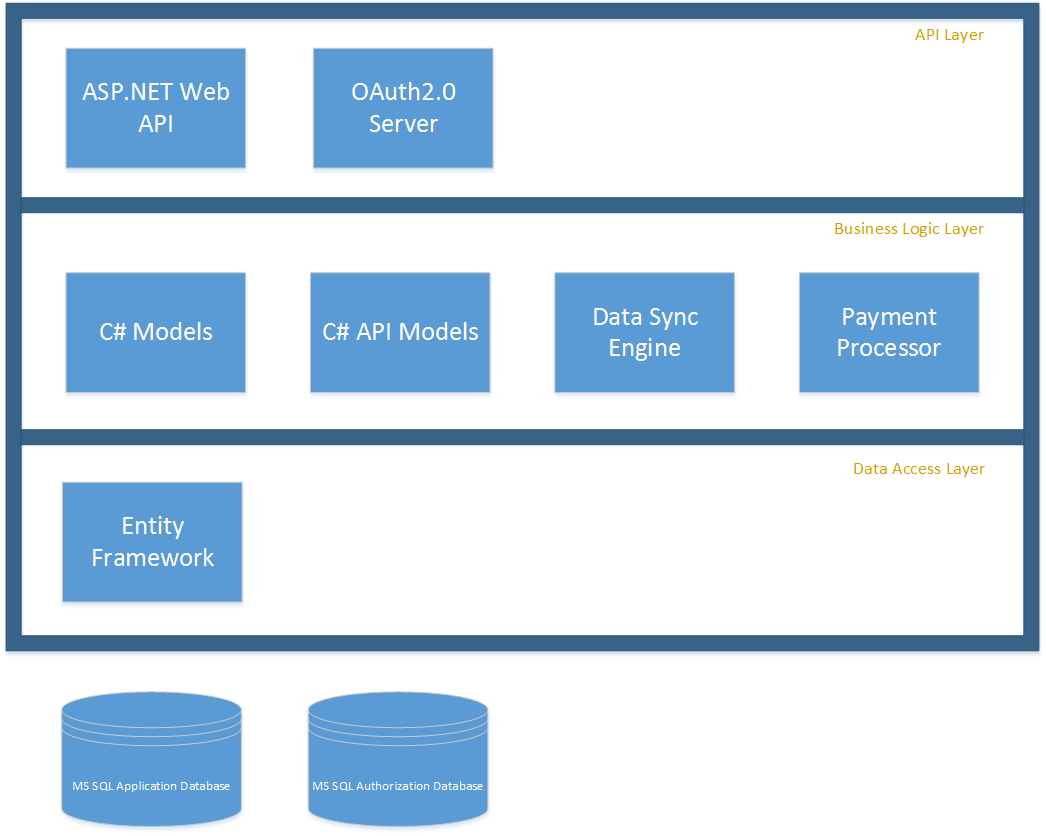
This document describes the high level architecture and design of Ciserro API. This is a preliminary design document. The concepts described here will be revisited and modified suitably during the development phase.

## Goals

* Develop an API for the generic care platform for Ciserro where care service providers from various segments can offer their services and will be a one-point stop for consumers to avail variety of care services.
* The API should be secured and exposed on HTTP protocol which could be easily be integrated with various client applications like web, mobile apps etc.

# Architecture Overview

## Layering Strategy



# Cross Cutting Concerns

The following Cross Cutting Concerns are addressed in this document:

* **Authentication**

The Authentication strategy is necessary for security. This strategy determines how to confirm that users are who they say they are.

* **Authorization**

The Authorization strategy determines user access to specific resources within the application.

* **Cryptography**

The Cryptography strategy is necessary for safeguarding sensitive information.

* **Configuration Management**

The Configuration Management strategy can make the application flexible and can reduce administrative overhead.

* **Caching**

The Caching strategy is necessary for the performance of the application.

* **Exception Management**

The Exception Management strategy is necessary for security and support of the application. A good strategy can help the user experience and can also reduce the complexity of troubleshooting problems in the application.

* **Auditing and Logging**

The Auditing and Logging strategy is necessary to identify issues and threats. A good logging strategy is helpful when troubleshooting the application.

* **State Management**

The State Management strategy determines the persistence of data that represents state during a session.

* **Validation**

The Validation strategy is necessary for validation of business rules and input.

* **Globalization and Localization**

The Globalization and Localization strategy is necessary to support other cultures and regions.

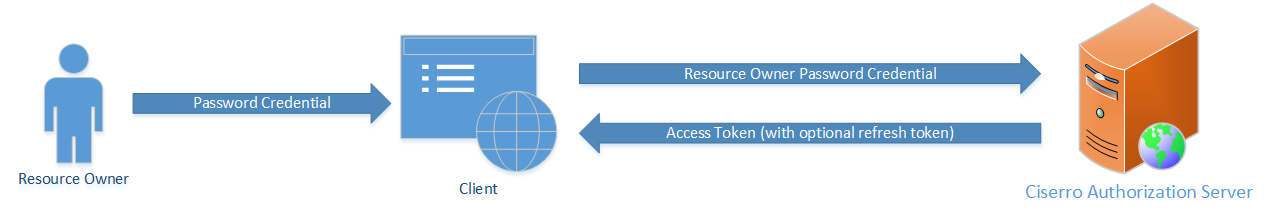
## Authentication/Authorization

### Ciserro OAuth Server

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. Ciserro will use a dedicated authorization server to authenticate/authorize the API requests. The consumer/client applications will have to request a JWT token from the Authorization server to be able to communicate with the API.

#### Access Token

The authorization server issues a JWT (JSON Web Token) to the client using Resource Owner Password flow. The following diagram illustrates the flow:



The user enters his/her username and password in the client. Client can then exchange the password credential for a short lived access token with the Authorization server. Once the token has been acquired, client can communicate with the API server to request for resources/API methods.

#### Refresh Token

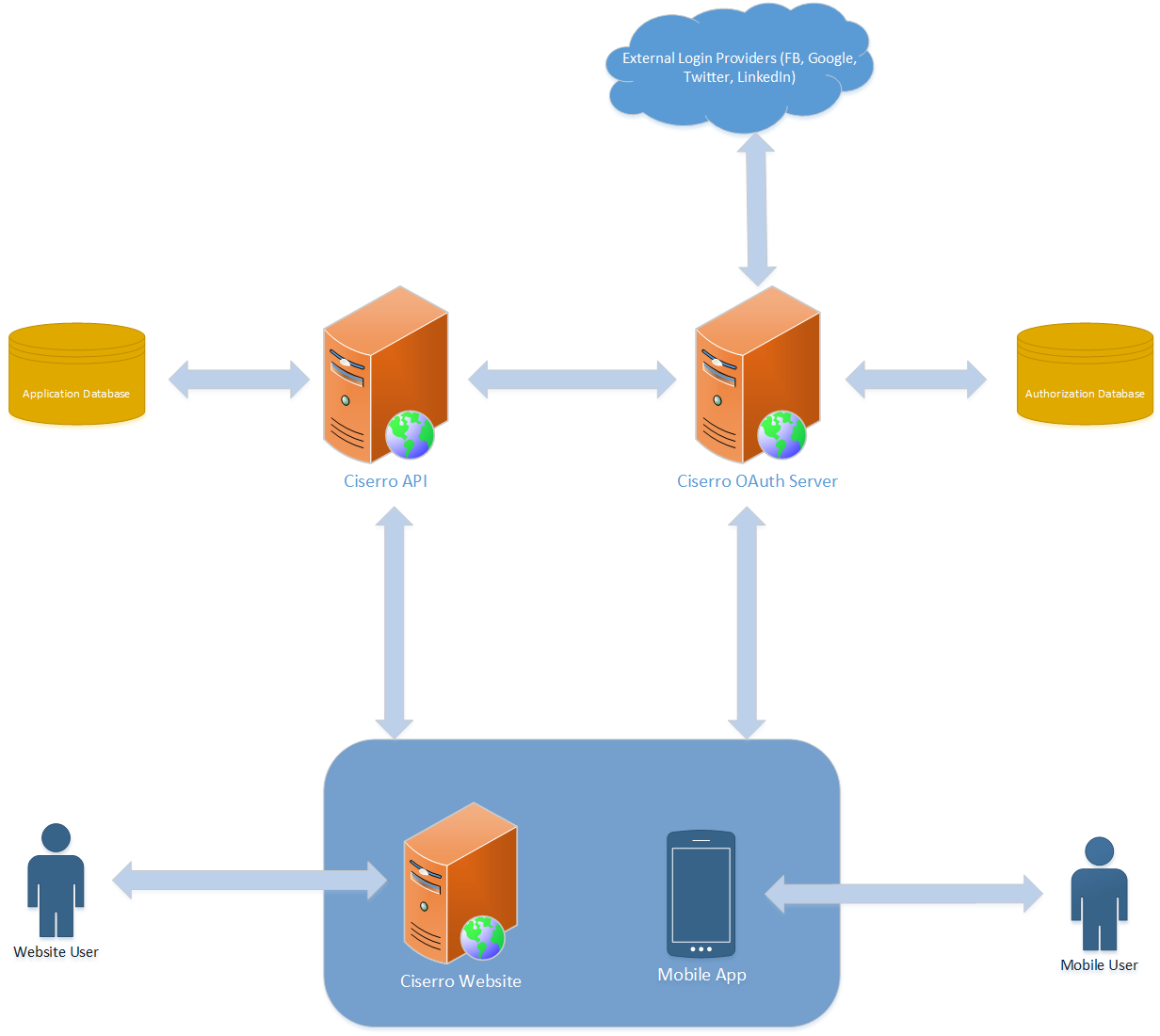
The access token issued by the Authorization server is short lived and expires within a configured time interval. As it is not desirable to prompt the user for his credential within that interval to retrieve a new access token, the server allows a way to refresh the access token/retrieve a new access token using Refresh Tokens.

The refresh token is not tied to an access token; it does not refresh an existing access token. The refresh token can be used to get a new access token without reinitiating the resource owner password credential flow.

#### External Login Providers

The Authorization server will be configured to connect to third party login providers like Facebook, Google, Twitter and LinkedIn. It will provide easy means using which the user can provide one of the mentioned social media accounts to register with the Ciserro application.

## Communication



## Cryptography

The user password will be hashed and stored in the database. No more sensitive data is being stored by this application; therefore, a cryptography strategy will not be necessary.

## Configuration Management

The web.config file will be used for connection strings and configuration settings.

## Caching

TBD

## Exception Management

Unhandled errors are handled by the OnException method in the base API controller.

## Auditing and Logging

Log4Net is used for logging information. By default, it is set to write to bin\logs folder in the root folder of the web API.

## State Management

There won’t be any session state used for the API. The user claims will be retrieved from the access token.

## Validation

There are two different ways in which validations are handled; Data access layer (Entity Framework) and the API layer validate the model that’s posted to the API methods before committing to the database.

## Globalization and Localization

This is out scope for the project. All the messages and constants will be stored in static classes/resource files for the ease of maintenance.

# API Layer

ASP.NET Web API 2 will be used to develop the API and Microsoft IIS will be used to host the service. This will be a RESTful API secured using SSL and OAuth2.0 bearer token authorization.

OAuth2.0 server developed using ASP.NET API 2 will handle authentication/authorization. This will be hosted as a separate website on IIS to off load the API server from authentication/authorization requests.

# Business Logic Layer

## Data Sync Engine

TBD

This section deals with migrating facility information from the government website to the Ciserro application database. This is likely to be a windows service to perform scheduled migration on a regular basis.

## Payment Processor

The following Stripe.NET library will be used for payment processing:

<https://www.nuget.org/packages/Stripe.net/>

# Data Layer

The data layer provides read/write access to the data store. Business logic should not be encapsulated in this layer.

## Entity Framework

The Entity Framework ORM will allow rapid development of domain models, therefore the Entity Framework Code First from Database approach will be implemented.

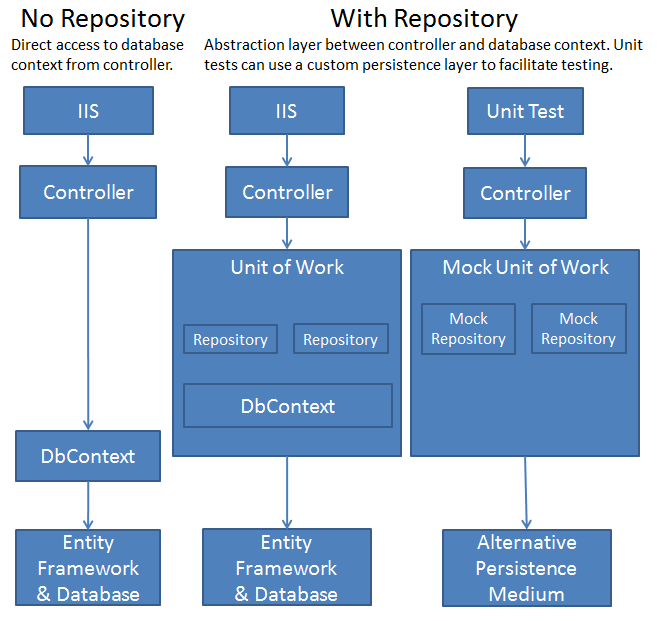
## Repositories

Following the Repository pattern, repositories for each entity is implemented via a generic interface for which a default implementation is provided.

## Unit of Work

A Unit of work class will be available to encapsulate Unit of Work logic utilizing the repositories and providing a single call to update the data store.

The unit of work – repository pattern will help in creating well defined abstraction layer between the API controller and the database context. The unit testing would become easy by mocking the unit of work and database context.



## Database

SQL Server 2012/’14 will be used for authorization and application databases.

# References

Microsoft Architecture Guide 2nd Edition

<http://msdn.microsoft.com/en-us/library/ff650706.aspx>