*Kiran Patel*

*travelMe*

A blogging application allowing backpackers to blog their travels across the world.

Last Updated: 28/09/2015

Table of Contents

[Summary 3](#_Toc428716403)

[Roles and Responsibilities 3](#_Toc428716404)

[Business Objects 3](#_Toc428716405)

[User 3](#_Toc428716406)

[Trip 4](#_Toc428716407)

[Post 4](#_Toc428716408)

[Media 4](#_Toc428716409)

[Object Relationships 4](#_Toc428716410)

[Architecture 5](#_Toc428716411)

[Methodology 5](#_Toc428716412)

[Documentation 5](#_Toc428716413)

[Naming Conventions 6](#_Toc428716414)

[Services 6](#_Toc428716415)

[Controllers 6](#_Toc428716416)

[Models 6](#_Toc428716417)

[Tools & Technologies 7](#_Toc428716418)

[Test Cases 7](#_Toc428716419)

[Quality and Standards 7](#_Toc428716420)

# Summary

TRAVELME is blogging application designed specifically for backpackers who are travelling the world. The application will provide the functionality to post blogs along with pictures about there current travels. Journeys will be organised into ‘Trips’ where each user may add as many trips as they wish. Within each trip there will be many posts. The web application will provide a way to store these posts and also view them for later viewing. Functionality to share with friends will also be considered. TRAVELME will use location data to track a person’s travels over a period of time, this will be later displayed through a mapping api such as Google Maps.

# Roles and Responsibilities

User can:

* Signs in
* Registers
* Add Trip
* Edit Trip
* Delete Trip
* Add Post
* Edit Post
* Delete Post
* Add Photo
* Remove Photo [Whilst adding]
* Remove Photo
* Share Trip
* Share Blog
* Set Location
* View Locations

# Business Objects

* User – Entity represents a user using the application
* Trip – Entity represents a trip which pertains to a user
* Post – Entity represents a post pertaining to a post will contain text and potentially images
* Media – Contains a media image where many may relate to one post

## User

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Notes** |
| ID | Uniqueidentifier | PRIMARY KEY NOT NULL |
| FirstName | Nvarchar(100) | NOT NULL |
| LastName | Nvarchar(100) |  |
| DateOfBirth | DATETIME |  |
| ProfilePicture | nvarchar |  |
| Email | Nvarchar(500) | NOT NULL |
| Password | Nvarchar(1000) | NOT NULL |

## Trip

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Notes** |
| ID | Uniqueidentifier | PRIMARY KEY NOT NULL |
| TripName | Nvarchar(20) | NOT NULL |
| TripDescription | Nvarchar(50) |  |
| TripLocation | nvarchar(75) |  |
| UserID | Uniqueidentifier | FOREIGN KEY CONSTRAINT TO USER |

## Post

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Notes** |
| ID | Uniqueidentifier | PRIMARY KEY NOT NULL |
| Post | Nvarchar(256) | NOT NULL |
| PostLat | Nvarchar(11) |  |
| PostLong | Nvarchar(11) |  |
| PostDate | datetime |  |
| TripID | Uniqueidentifier | FOREIGN KEY CONSTRAINT TO TRIP |

## Media

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Notes** |
| ID | Uniqueidentifier | PRIMARY KEY NOT NULL |
| Media | Nvarchar(255) |  |
| PostID | Uniqueidentifier | FOREIGN KEY CONSTRAINT TO POST |

Uniqueidentifier is used as a primary key data type instead of INT, this is because the entities will use GUIDs. This decision was made to ensure that each record in the application was unique.

The Decision was made to use nvarchar instead of varchar because nvarchar can store unicode data whilst varchar is restricted.

# Object Relationships

* Each User (1:1) has no or many Trips (0:\*)
* Each Trip (1:1) has no or many Posts (0:\*)
* Each Post (1:1) has no or many Media (0:\*)

# Architecture

TRAVELME will use an n-tier architecture along with the Model-View Controller design pattern. The main layers will be the presentation, business logic and data access layer. These layers will however be broken down further using the Repository Pattern and with the use of services. The application will also make use of dependency injection. The decision was made to develop an n-tier architecture so that the application could be made as maintainable as possible and to make it testable.

Layers:

* SQL Server
  + Stores data with a table for each entity
* Data Access
  + Interacts with the database to execute CRUD functions. This is done through Object-Relational Mapping (nHibernate). The nHibernate By Code mappings will be used to ensure in code mapping without xml configurations
* Repository
  + Interacts with the data access layer where there is a Generic Repository and all entities will use this repository to access the database.
* Service Layer
  + Contains all the business logic in order to make the controller thin. Service layer will interact with the repository.
* Controller <-> Model
  + Controller handles user input and output, Controller will have minimal logic as most will be in service.
  + Models are the models of the application to represent entities.
* View
  + Display cshtml to the user
  + Repeatable sections such as the Post will be made into partials within the presentation layer. These reusable sections will be used to display posts

Each Layer will be split into different projects with the exception of the Model, Controller and View which will reside in the same project for the web project.

# Methodology

The project will be developed using a Test Driven Development (TDD) methodology. The use of frameworks such as nUnit and Moq will be used to test the functional requirements of the project. The project will aim to have >90% code coverage.

After each unit of the application is written, unit tests will be used to see if it works as functioned.

# Documentation

The project will be documented through documents such as these like a specification. In code the plugin Stylecop will be used to thoroughly document the project. Changes to the database will be documented on a sql script within the dataaccess project.

## Naming Conventions

Project Files – com.kiransprojects.travelme.<ProjectName>

Unit Test Project - com.kiransprojects.travelme.<ProjectName>.<Tests>

Repository - <RepositoryName>Repository

Services - <ServiceName>Service

Factories - <FactoryName>Factory

# Services

Services will be used to allow controllers to be thin in terms of business logic as possible. Services will interact with the Data Access layer in order to interact with the database.

* PasswordService
  + Responsible for ensuring passwords are verified and hashed
* LoginService
  + Responsible for verifying a user has passed correct credentials
* RegisterService
  + Responsible for registering a user
* UserService
  + Responsible for interacting with the UserEntityRepository
* TripService
  + Responsible for interacting with the TripRepository
  + Can get location data from the Post entity for a specific user
* PostService
  + Responsible for interacting with the PostRepository
* MediaService
  + Responsible for interacting with the MediaRepository
* LoggerService
  + Responsible for logging

# Controllers

Controllers will receiver user input and send a response. This component is in charge of receiving input, communicating with the necessary models and Services and returning a view.

* HomeController
  + Index
* AccountController
  + SignIn
  + Register
  + ForgotPassword
* UserController
  + Index (with ID)
  + EditUser
  + DeleteUser
  + GetAll
* TripController
  + Index(all users trips)
* PostController
* MediaController

# Models

Models will be used to on the containers of business entities. Business entities will be encapsulated in models to be used by the controller.

* User View Model
  + Holds User Entity and Feedback Entity
* Location
  + Holds Location data without entire post

# Tools & Technologies

ASP.NET MVC

C#

Visual Studio 2013

SQL Server 2012

nHibernate (Object-Relational Mapping)

nUnit (Unit Testing)

Moq (Mocking Framework)

StyleCop (Documentation)

Ninject (Dependacy Injection)

LINQ

# Test Cases

|  |  |  |
| --- | --- | --- |
| Repository |  |  |
| GetByID\_ExistingEntity |  |  |
| GetByID\_NonExistingEntity |  |  |
| GetByID\_NullEntity |  |  |
| Update\_ExistingEntity |  |  |
| Update\_NonExistingEntity |  |  |
| Update\_NullEntity |  |  |
| Update\_InvalidEntity |  |  |
| Insert\_ExistingEntity |  |  |
| Insert\_NonExistingEntity |  |  |
| Insert\_NullEntity |  |  |
| Insert\_InvalidEntity |  |  |
| Delete\_ExistingEntity |  |  |
| Delete\_NonExistingEntity |  |  |
| Delete\_NullEntity |  |  |
| InsertSubList\_ExistingEntity |  |  |
| InsertSubList\_NonExistingEntity |  |  |
| InsertSubList\_NullEntity |  |  |
| InsertSubList\_InvalidEntity |  |  |

# Quality and Standards

Quality of the code will be deduced from various factors. Firstly, the use of unit tests will ensure that the system provides the expected behaviour of the system. Visual Studio tools will also be used to ensure a good quality. Tools such as Code Analysis and Code Metrics will be used to determine the quality.