\\The database scripts for this application is in DomainMapping/DDL.sql

The app conforms well to DDD, it uses [Rich Domain Model](http://www.anicehumble.com/2013/12/domain-modelling-aka-object-oriented-programming.html) instead of [Anemic Domain Model](http://www.martinfowler.com/bliki/AnemicDomainModel.html)

The solution has [three main layers](http://geek-and-poke.com/geekandpoke/2013/7/13/foodprints) only. RideOfYourLife (front-end) + Marshaller(middle-tier, just a DTO marshaller) + Domain(contains the domain model with behavior)

Domain project contains the domain models, POCO objects. The model contains the behavior

DomainMapping project constains the mapping between the Domain models and relational

Domain. Test project test the Domain’s behavior

Dto project contains the *user-facing* model*,* using DTOs avoids the [Stripper Pattern](http://ayende.com/blog/4011/the-stripper-pattern) problem

[Marshaller](https://plus.google.com/+AngularJS/posts/ANhZTovQLR6) project (which is usually named a service project) is named as such to emphasize that the work of this layer is to handover DTOs to and from the front-end, the business logic should [not](http://www.martinfowler.com/bliki/AnemicDomainModel.html) happen at the service layer, it should happen at the domain model. The service (e.g., ASP.NET Web API, WCF, ServiceStack) should just marshal data, and should be devoid of business logic. This layer is made in ServiceStack, will learn ASP.NET Web API soon. This layer is made in ServiceStack, it’s [quite fast](http://www.ienablemuch.com/2013/08/servicestack-fast-service-framework.html) and has a consistent design pattern

RideOfYourLife project is the front-end, it uses AngularJS

For pragmatism purposes, though Domain project should only handle business logic, it has a reference on ORM, not exactly for persistence concerns, it’s more on to facilitate efficient fetching strategies and optimization