# Minimum Requirements

.net core 3.1

.net standard 2.1

SQL Server 2014

# Integration Test Requirements

SQL LocalDb with MSSQLLocalDB database

Chrome 80

# Application Goal

1. View a book
2. Search for books based on the book title, author, series, and rating
3. Delete a book
4. Add a book
5. Update a book

The integration tests are written using Gherkin which is an engli-ish type language. The goals are written more in depth in the gherkin feature files located in: [project root]\tests\integratin\Katz.Web.Website.Tests.Integration\Features and have a .feature extension

# Tested With

The application was tested with the Chrome 80 browser.

# Decisions

I started building the application using TDD, however due to time constraints I became unsure if I could finish it in time and thus I skipped doing unit tests first. I still continued to write integration tests first in order to make sure I met the requirements outlined to me. This resulted in a lacking of unit tests and coupling of individual units behavior.

I wrote the code as a proof f concept application using the MVC website template and thus would make different decisions in code and UX if this was for production. A UX example is when displaying search results there is no paging and it would be needed since the results could become massive. A code example is that there are no null checks and thus if the code is re-used in a different context then the consumer could receive obtuse exception without ensuring their arguments are not null. It would also make bug hunting harder. I am a fan of C#’s new nullable reference types to help with this problem.

I layered the code into a UI layer(the Katz.Web.Website project), the business layer(the Katz.Core project), and a data layer(the Katz.Data project). The UI layer deals with rendering the display to the user. It has models which are just DTOs with data annotations. The business layer deals with the creation and processing domain objects and rules. The data layer uses entity framework to get data from and to a data store that the domain layer uses to apply business rules. The domain layer’s data objects have constructors to create valid business objects. For example, you cannot create a Book without BookDetail and you cannot create a BookDetail without a title.

# Running The MVC Project

Each time the project runs in development mode it will drop and create a new database with seed data found in the Startup class. To disable this feature comment out the line “InitializeBooks(app);”. The connection string for the database is located in “appsettings.json” and points to the MSSQLLocalDB database which is a default database that comes with LocalDB. Assuming you already have it then you should be able to just run the project out of the box. The connection string can be changes to another SQL instance of your choice such as a SQL Server Express 2014 instance. A SQL Script to create the database is included in the [Project Directory]\scripts directory.

# Final thoughts

I’m happy to talk about the code and my decisions more in depth and am looking forward to the on-site visit.