**Quest Engine**

**Requirements**

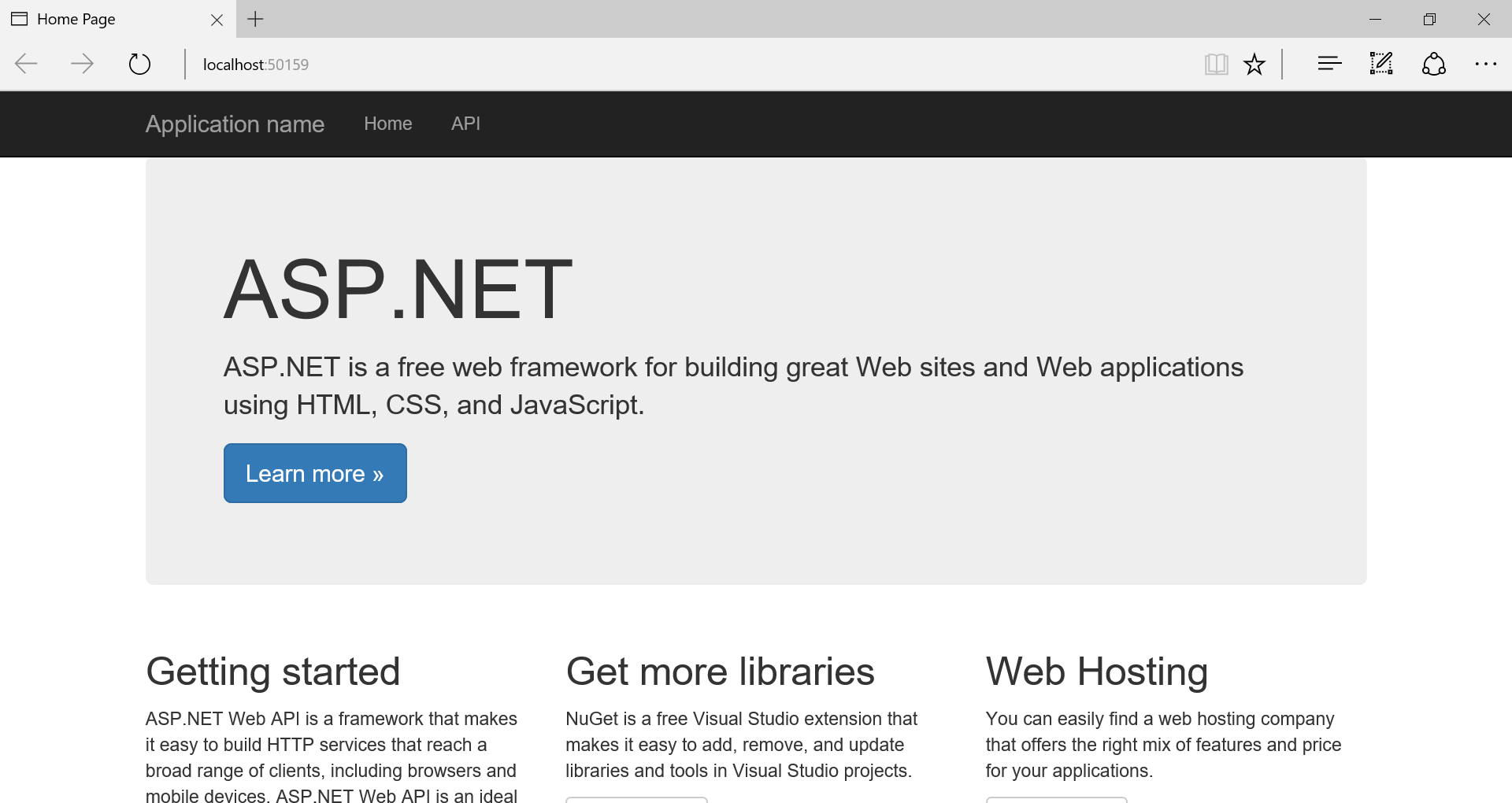
Required to compile the solution

* Visual Studio 2017 Community or equivalent
* .NET Framework 4.6.1
* IIS Express to load QuestEngine project
* Google Flatbuffers for Flatc.exe
  + Not needed unless you wish to re-compile the flat buffer file
  + Pre-compiled already done

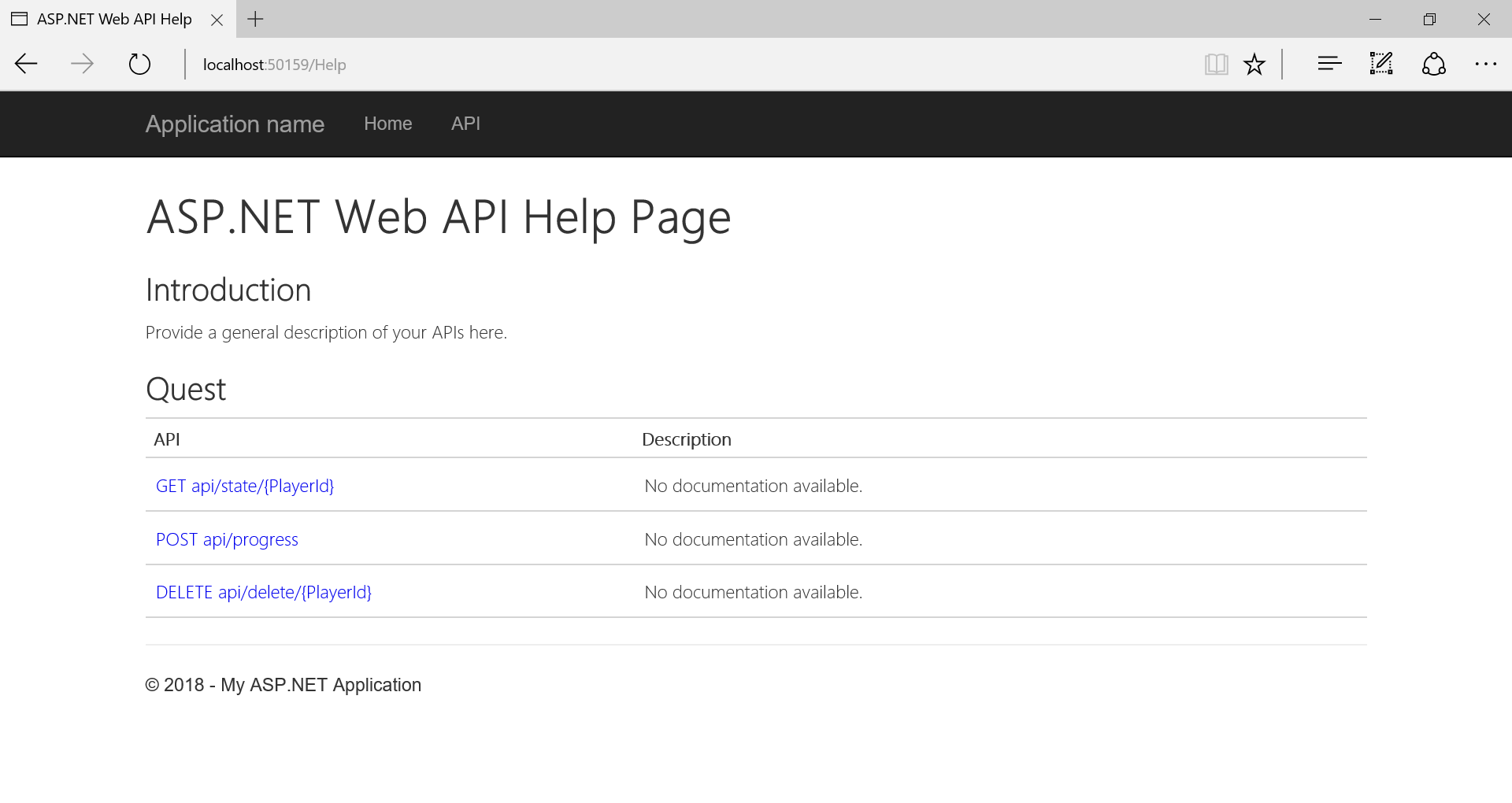
**Solution**

Compile Solution and Test Run in MSUnit

1. Open up **QuestEngine.sln**
2. Select **Debug** (or Release) and compile entire project
3. Hit **CTRL-F5** to run solution directly on local machine
4. The browser should popup with the standard Web API home screen



1. Navigate to API page to take a look at exposed APIs
   1. Sample: <http://localhost:50159/Help>

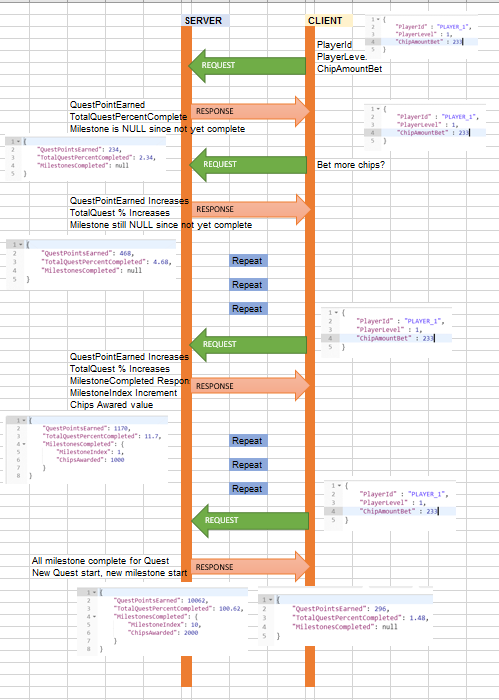


1. You can test the API
2. With frontend testing, prefer to use POSTMAN or FIDDLER
3. Make State call
   1. Sample: GET http://localhost:50159/api/state/PLAYER\_1
4. Make Progress call
   1. Sample: POST http://localhost:50159/api/progress
5. Delete a user/player from system
   1. I added this myself just for testing purposes
   2. Sample: DELETE http://localhost:50159/api/delete/PLAYER\_1
6. URL/Body, use exactly as describe in manual file

Testing with MSUnit

1. Compile **QuestEngine.Tests**
2. Run the Test Explorer
3. Select and run all test
4. Most comprehensive test is **Test\_6\_Simulate\_Quest\_Multiple**
   1. This test simulates 3 users questing at the same time
   2. It tests everything from API **/State** and **/Progress**
   3. Simulating reaching milestones and quests
   4. This is directly calling the internal implementation, not the Façade
5. Simple testing directly from Web API layer is provided through **QuestControllerTest**
   1. Not much testing at this level, prefer to use the one above

**Milestone Sequence Diagram**



**Quest Configuration JSON Setting**

File: **QuestConfig.json**

|  |  |
| --- | --- |
| **Property** | **Description** |
| RateFromBet | Configurable to calculate Quest Point Accumulation |
| LevelBonusRate | Configurable to calculate Quest Point Accumulation |
| Quests | Array of N Quests can have as many quests as needed |
| Quests.PointsCompleted | Points required to complete this quest |
| Quests.MileStones | Array of N Milestones within Quest, as many as needed |
| Quests.MileStones.PointsCompleted | Points required to reach this milestone |
| Quests.MileStones.ChipsAwarded | Chips awarded for reaching this milestone |
|  |  |

**Player Data saving using Flatbuffers**

File: **PlayerSchema.fbs**

The file used by Flatbuffers **FLATC.EXE** to auto generate PlayerSchema.cs file. The standard use to save/load game files for fast optimized serialization and transport.

|  |  |
| --- | --- |
| **Property** | **Description** |
| Player | Root type, player data |
| Player.name | Player Name |
| Player.quest | Current quest player is on |
| Player.milestone | Current milestone player is on |
| Player.pointsearned | Total points earned for current quest |
|  |  |

File: **PLAYER\_NAME.bin**

Currently saves player data in **Players** folder, with each file corresponding to player name, such as PLAYER\_X.bin

**Improvements and Known Issues**

[IMPROVEMENT]  
Get rid of Flatbuffers and use a proper storage such as DB or NoSQL DB to save game data. I use flat buffers for this exercise, so I can learn the library. (Please forgive me)

[IMPROVEMENT]  
Perhaps refactor and decouple more a bit from Quest and Milestone, each having their full-blown class file so as to support more complicated quest and milestone (similar to RPG games with complicated quest)

[IMPROVEMENT]  
Proper action filters setup, extending from ActionFilterAttribute, to support global exceptions, logging, authorization, etc.

[IMPROVEMENT]  
If this wasn’t turned based but real time based, perhaps switch to WebSockets, SignalR, message queue, asynchronous APIs, or a combination of them.

[KNOWN ISSUES]  
Player naming issues, can’t guarantee support with weird character names, capitals, symbols, etc.

[KNOWN ISSUES]  
Quest config issues with respect to input, only supports ‘good’ data, program may behave strange if ‘limits’ are broken in input or setting, such as large bet size that overflows quest/milestone completion targets, milestone and quest completion points mismatch, data inputs such as rates, etc.

[KNOWN ISSUES]  
MSUnit test may fail if certain procedure is run before others, such as getting StateResponse before ProgressResponse, due to unit test actively checking for return value > 0. Will need to reorganize this a bit to be fully unit testable.