**Online Task List Design Choices**

**First big choice: What sort of Web Interface**

My options here are Web Forms, MVC, or Web API (As I’m using .NET Framework and not Core). I’ll be going down the MVC route. Web Forms are a little outdated now, and as my solution isn’t going to be too driven by external API calls (Bar AJAX calls when creating/ticking off a taskItem) I don’t need to worry about opening up to any sort of integration. It was a toss-up between Web API and MVC, as I’m seeing more and more developers trying to move away from MVC – Reason being MVC’s will include C# code being written in your HTML views, which isn’t as clean as people would like. It also calls into play SOC. I’ve decided to go with MVC as it was mentioned in the job description (And I’ll probably learn something new on the journey!).

**Second big choice: Code First method v Database First**

I’ve gone for a Code First approach. This is for a number of reasons:

* Time constraints - Using Migration I can get VS to write most of the table creation code for me.
* GitHub knowledge – I don’t have much experience with GitHub repositories outside of reading documentation/RESTful API uploaded to it or downloading singular files, which means I don’t know how well me writing my code for my local MySQL instance is going to translate. Using Code First should ensure that when the solution is built on your end, you end up with a replicated schema built the same as mine.
* Security – There’s some excellent advice out there that says ‘If you can possibly not handle passwords yourself, do that’. By including VS ‘Individual User Accounts’ authentication, I can eliminate the time needed to write code to handle the unsafe password string, salt it, hash it (iterating n times), and compare it.

Conversely, if I were to go for Database First, I would be more on home territory (As this is what my previous projects have been using) – and find it easier to maintain SOC, for example, my SQL statements in a class library ‘OnlineTaskList.Core’ and the web interface on ‘OnlineTaskList.Web’. In terms of security, for authentication I would use OAuth2 and JWT (Time dependent).

**Creating my Controller from template**

I’ve gone with the EntityFramework template, as this will take care of a lot of the task list functions you’d expect to see (CRUD). As I’m using Migration (NuGet), I would need to look into how I go about retrieving my data (If it’s any different to using the connectionstring in my Web.config) were I building a controller from an empty template. Again, due to time constraints I’m going to take the easier approach.

**Quick note on performance**

Because I’ve opted for a few ‘auto code generating’ methods, I have a lot of junk thrown into my code as well as the stuff I actually need. For example, when letting VS handle my user accounts, I’m going to get a lot of C# and even table structure relating to things I don’t intend to use for this simple project, such as email addresses, password resets/updates etcetera. This is all stuff my compiler is needing to build, which only stands to increase the size of the project without increasing the functionality. I will leave ‘cleaning up’ as much of this as I can to my final step, as my time limit leaves me with having to prioritise actually having a complete project, security, and performance.

**Architecture**

As mentioned above, were I able to allocate more time to this I would be approaching this with SOC more in mind, for example by separating my files into ~.Web and ~.Core - .Web would hold my files specifically pertaining to the Web interface, and .Core would be my class library containing my SQL statements, my repositories, interfaces, etcetera.

**Security**

For the most part, I avoided the ‘bigger’ security risks – Namely losing sensitive data through writing code to handle passwords myself. By leaving a lot of that side to VS to auto-generate for me, I can skip past the usual pitfalls of SQL injection. With the exception of the ‘Completed?’ tickbox, I can also leave XSRF protection to the auto-generated code.

In terms of protected functionality, I have set up a basic redirect should someone attempt to view a task list without being logged in. However – I ran out of time before I got to adding user protection in terms of allowing a user to edit someone else’s task if they know the Id. I would approach this via an if statement that checks the user the task belongs to against my GetCurrentUser() method.

**Performance**

This is where my trade-off of increased security and lower time usage comes back to bite me. By installing and utilising (what is essentially 5% of the potential) EntityFramework and User Accounts handling, the solution is in total an awful lot larger than it would be if I just had a couple of SqlHelper classes in a ~.Core.

However, I’m able to increase performance as far as web request size and user experience is concerned by using partial views and AJAX requests to avoid reloading entire pages whenever the user wants to tick a task off, or create a new one.

**Time Saving / Easy Approaches**

* Migration to auto-create db tables
* Visual Studio Authentication to remove the need of password handling
* EntityFramework controller template to quickly get the scaffolding of my CRUD actions

**Things you should know**

For any points that I have taken the ‘Time save / Easy approach’ that you feel I haven’t demonstrated enough of an understanding of the alternative in my explanation, please feel free to query these with me further to understand where I am with the subjects.