**Ideas for future modifications**

**Description**

The reason I created this document is that it’d be bad to put the real minimum requirements and not important ideas all together. So non-functional requirements were moved here, ideas for modifications and ideas what it’d be interesting to try on the project are described here also. I’ve done it just not to lose my focus, so I could implement the most important requirements in the first place.

In the code I’ll put #BAD hashtags and some description so as anybody could see places that require refactoring and could understand why.

**Non-functional requirements**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | | **Description** | |
| User interface | | Dashboard | Rating tables are located on the separate tab. But instead of this it’s better to try to implement Stores pattern with Navigation and all this stuff. |
| Architecture | | The ability to easily add new functionality and remove the old one.  Low interdependence and strong ties.  We are developing the architecture in such a way that in the future it is possible to easily replace one place of data storage (e.g., JSON files) with another (e.g., database), with the ability of dynamically changing the number of rating tables in the event that we add game difficulty levels in the future, with the easy ability of adding new game difficulty levels. | |
| Speed, efficiency and system load | | * unnecessary data should not be uploaded; * there should not be a bunch of redundant intermediary classes; * fast dashboard generation; * fast uploading of data. * should be all ***async*** so the main thread doesn’t get blocked when saving or loading files. | |
| Data storage | | * unnecessary data is not stored; * no more than 10 files either for the rating table or for save points; * there must be cache for savepoint, so only 3 savepoint must be loaded in program at the same time; * rating tables of each difficulty are loaded in accordance with which table the user has switched to. After loading, if the rating table of this difficulty has not been viewed for 5 minutes, it is unloaded until the user goes to it again.   Useful services: ICacheService, MemoryCache | |
| Technology stack | | Visual Studio 2022, .NET 8.0, WPF | |
| Hardware requirements | Processor | Intel ® Core ™ 2 / 2 Duo / Pentium ® /  Celeron ® / Xeon™ / i3 / i5 / i7 чи AMD 6  / Turion ™ / Athlon ™ / Duron ™ /  Sempron ™ with clock frequency no  lower 1.5 GHz. | |
| RAM | At least 2GB of RAM is recommended | |
| Hardware architecture | — 32-bit (х86);  — 64-bit (х64). | |
| Target platform | | Windows 10 | |
| Stability | | If the system shuts down unexpectedly or the user hard closes the program, data is automatically saved. | |
| Security | | Limit the user's ability to edit saved files directly through the file explorer. | |
| Usability | | There should be data deletion protection in the program directly. For example, dialog boxes that ask for user permission.  Since loading and saving data must be asynchronous, when data is being saved, user ***mustn’t be*** allowed to change it!  Also, if the user manually changed files through the File Provider, then since we are not be able to upload data, a message about this is displayed. | |
| Maintainability | | There should be logger. | |
| Localization | | Ukrainian and English languages | |

**Userful links:**

<https://learn.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions>

<https://www.youtube.com/watch?v=zlnq3HAhwr8>

<https://www.google.com/search?q=cache+C%23&oq=cache+C%23&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIHCAEQABiABDIHCAIQABiABDIHCAMQABiABDIHCAQQABiABDIHCAUQABiABDIHCAYQABiABDIHCAcQABiABDIHCAgQABiABDIGCAkQLhhA0gEJMTA1MDlqMGoxqAIAsAIA&sourceid=chrome&ie=UTF-8&safe=strict>

<https://learn.microsoft.com/ru-ru/dotnet/core/extensions/caching>

**Ideas for code refactoring or upgrading:**

|  |  |
| --- | --- |
| **Name** | **Description** |
| “Factory method” | In the future it’s planned to add an ability to work with DB, maybe CSV, maybe another kind of storage. So it’d be great if system allowed us easily to do it and delete a tool for working with JSON-files easily.  It’s important to mention, that it’s probably better to make possible to pass parameter of type T (any type). It’ll be done so as only some classes could set, for example, filepath, if we’re talking about CSV-files. As I See it now:  interface IXXXFactory<T>  {  IXXX Create(T parameter);  }  class CSVFactory : IXXXFactory<string>  {  IXXX Create(string filepath);  }  Not sure how I’ll do it, just not to forget the idea. |