

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Computer Graphics and Interaction**

Using Game Engine for Note Taking and Sharing

Ivan Oryshchenko

**Supervisor: doc. Ing. Jiří Bittner, Ph.D.
May 2023**

I. Personal and study details

Student's name: **Oryshchenko Ivan** Personal ID number: **490854**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Computer Graphics and Interaction**
Study program: **Open Informatics**
Specialisation: **Computer Games and Graphics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Using Game Engine for Note Taking and Sharing

Bachelor's thesis title in Czech:

Využití herního enginu pro vytváření a sdílení poznámek

Guidelines:

Bibliography / sources:

- [1] Nicholas J Cepeda, Harold Pashler, Edward Vul, John T Wixted, and Doug Rohrer. Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychol Bull*, 132(3):354–380, May 2006.
- [2] John Dunlosky, Katherine A. Rawson, Elizabeth J. Marsh, Mitchell J. Nathan, and Daniel T. Willingham. Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest*, 14(1):4–58, 2013. PMID: 26173288.
- [3] Hermann Ebbinghaus. Memory: a contribution to experimental psychology. *Ann Neurosci*, 20(4):155–156, October 2013.
- [4] Jeffrey D. Karpicke and Janell R. Blunt. Retrieval practice produces more learning than elaborative studying with concept mapping. *Science*, 331(6018):772–775, 2011.
- [5] Jaap M. J. Murre and Joeri Dros. Replication and analysis of ebbinghaus' forgetting curve. *PLOS ONE*, 10(7):1–23, 07 2015.
- [6] Joseph Novak and Alberto Cañas. The theory underlying concept maps and how to construct them. 01 2006.
- [7] Joseph D. Novak, D. Bob Gowin, and Jane Butler Kahle. *Learning How to Learn*. Cambridge University Press, 1984.
- [8] Doug Rohrer and Hal Pashler. Increasing retention without increasing study time. *Current Directions in Psychological Science*, 16, 08 2007.

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Jiří Bittner, Ph.D. Department of Computer Graphics and Interaction

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **17.02.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Ing. Jiří Bittner, Ph.D.
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

Acknowledgements

To my parents, thank you.

Declaration

I hereby declare that I have completed this work independently and have quoted all the referenced literature.

Abstract

This thesis is about the design and development of a note-taking app that combines effective studying techniques, gamifies the study process, and lets users freely explore each other's notes.

Here I summarize research behind effective learning, analyze design decisions of different note-taking apps, introduce the technical tools used to build the project and explain the code architecture and logic. Then I test the application on real users, collect feedback, and make changes based on it. In the end, I outline the future prospects of the project.

Keywords: note-taking, active recall, spaced repetition, mind-maps, unity

Supervisor: doc. Ing. Jiří Bittner,
Ph.D.
Praha 2, Karlovo náměstí 13

Abstrakt

Tato diplomová práce se zabývá návrhem a vývojem aplikace pro psaní poznámek, která kombinuje efektivní techniky výuky, gamifikuje proces studia a umožňuje uživatelům volně prozkoumávat poznámky ostatních.

Shrnuji zde výzkumy, které stojí za efektivními studijními technikami, analyzuji návrhová rozhodnutí různých aplikací pro psaní poznámek, představuji technické nástroje použité k vytvoření projektu a vysvětluji architekturu a logiku kódu. Poté aplikaci testuji na skutečných uživateli, sbírám zpětnou vazbu a na jejím základě provádím změny. Na závěr nastíním průběh budoucího vývoje projektu.

Klíčová slova: psaní poznámek, aktivní vzpomínání, rozložené opakování, myšlenkové mapy, unity

Překlad názvu: Využití herního enginu pro vytváření a sdílení poznámek

Contents

1 Introduction	1	3.4.7 Search implementation.....	25
2 Theoretical background	3	3.4.8 Review priority algorithm ...	26
2.1 Learning theory	3	4 Tests and feedback	27
2.1.1 Active recall.....	3	4.1 Observations during testing	27
2.1.2 Forgetting curve	3	4.2 Suggestions received	28
2.1.3 Spaced repetition	4	4.2.1 Feedback discussion	29
2.1.4 Concept mapping	5	5 Discussion and limitations	31
2.2 Design analysis	5	5.1 Lacking features	31
2.2.1 Alternative to second brain...	6	5.2 Optimization	31
2.2.2 Generic vs Specific use case ..	6	5.3 AI applications	32
2.2.3 Tag system vs File explorer ..	6	5.3.1 Trainable spaced repetition	
2.2.4 Spatial file explorer	6	algorithm.....	32
2.2.5 Article vs Card note format ..	7	5.3.2 Semantic search.....	32
2.2.6 Rigid vs Free spaced repetition		5.3.3 Autocomplete questions.....	32
schedule	8	5.3.4 Check answers with AI	33
2.2.7 Private vs Public notes	8	5.3.5 Recommendation system....	33
3 Implementation	11	6 Conclusion	35
3.1 3-layer architecture	11	Bibliography	37
3.2 Tools.....	11	A Contents of electronic appendix	39
3.2.1 C Sharp	11	B User manual	41
3.2.2 Unity.....	11	B.1 Desk view	41
3.2.3 OpenAPI	12	B.2 Note view	42
3.2.4 Google Cloud Run	12	B.2.1 Note view edit mode	42
3.2.5 MongoDB.....	12	B.2.2 Note view review mode.....	42
3.3 Frontend	12		
3.3.1 Player class	12		
3.3.2 DeskItem, Node and Folder			
classes	13		
3.3.3 Shortcuts	15		
3.3.4 Level of Detail.....	15		
3.3.5 NoteView and ReviewBoxes			
classes	18		
3.3.6 GameModeManager class ...	19		
3.3.7 SaveManager class	20		
3.3.8 ZoomController class	20		
3.3.9 Object pools	21		
3.3.10 Search bar	21		
3.3.11 Logging	22		
3.4 Backend	23		
3.4.1 Layered backend architecture	23		
3.4.2 Session vs Token			
Authentication	24		
3.4.3 Registration flow.....	24		
3.4.4 Read/write permission.....	25		
3.4.5 Data Transfer Objects	25		
3.4.6 Rate limiting	25		

Figures

Tables

2.1 Normalized saving scores (the amount of information that was retained) plotted for Ebbinghaus, Mack, Seitz, and Dros[6].	4
2.2 Concept map example by Dr. Ali Abdaal.	5
2.3 Examples of note-taking apps using tags and file explorer	7
2.4 Visual comparison of article and card note format	8
3.1 3-layer web app architecture. . .	11
3.2 Desk example.	13
3.3 Shortcuts.	15
3.4 Open folder.	16
3.5 Closed folder.	16
3.6 Edit mode.	17
3.7 Review mode.	17
3.8 Old note view.	18
3.9 Finite State Machine at the core of the Unity project	19
3.10 Desk GET endpoints.	21
3.11 Layered backend architecture.	23
3.12 Endpoints responsible for authentication and authorization.	24
3.13 Endpoint using DeskDTO for updates	25
4.1 Example of a desk filled with subjects.	27



Chapter 1

Introduction

Over the years of using note-taking apps, I've accumulated many ideas on ways to improve them, so much in fact that there were enough to make a whole app around them. And most of these ideas stem from the root reason that, like many people, and especially students, I take notes to learn new information.

My problem is then that note-taking apps are implicitly designed to be used, that is the more you use them, the more dependent on them you become. This happens because every time you want to recall, for example, a lecture, you have to find it in your notes. And the number of times this happens scales almost 1-to-1 with the number of notes you take. With this app, I want to break this cycle by making the user use the app less and less over time.

How to achieve this? Well, both old and new research points us to a set of high-utility learning techniques that result in amazing retention of information with minimum time investment. These are active recall, spaced repetition, and mind maps. They all work towards creating a deeper understanding of the material in the learner and signal to the brain to store the information in long-term memory.

Besides these techniques, there are two other ideas I want to bring to the note-taking apps genre. First, make notes public so that any user can freely explore the notes of any other at any time. Second, make the user experience more interactive and alive, thus a game engine was chosen to build the frontend of the application.

In summary, this app will combine multiple effective learning strategies in one application, make the study process more fluid, and let users freely explore each other's notes.

Chapter 2

Theoretical background

2.1 Learning theory

This section discusses some of the lesser-known learning techniques and research supporting their effectiveness.

2.1.1 Active recall

Active recall is a learning strategy that involves actively recalling information from memory, rather than simply reading or reviewing it. This can be done through various methods such as quizzing oneself, writing out the information, or discussing it with others. Research has consistently demonstrated that incorporating active recall into one's study routine can lead to significant improvements in learning outcomes.

One study conducted by Karpicke and Blunt[5] compared the effects of active recall with those of rereading on learning. Participants were given text materials to study and were then tested on their recall of the information. The results showed that active recall was significantly more effective in improving learning outcomes than rereading. This finding has been supported by Dunlosky, et al.[2].

The effectiveness of active recall may be due to the fact that it requires greater effort and cognitive processing, signalling the brain to absorb the new information quicker.

2.1.2 Forgetting curve

The forgetting curve is a theory in psychology that describes the decline of memory retention over time. The original formulation of the curve was proposed by Hermann Ebbinghaus[3] in the late 1800s, who found that memory retention significantly decreases over time unless effort is made to retain the information.

Since Ebbinghaus' initial study, numerous research papers have investigated the forgetting curve and have found similar results. For example, this study[6] from 2015 successfully replicated the original findings (figure 2.1).

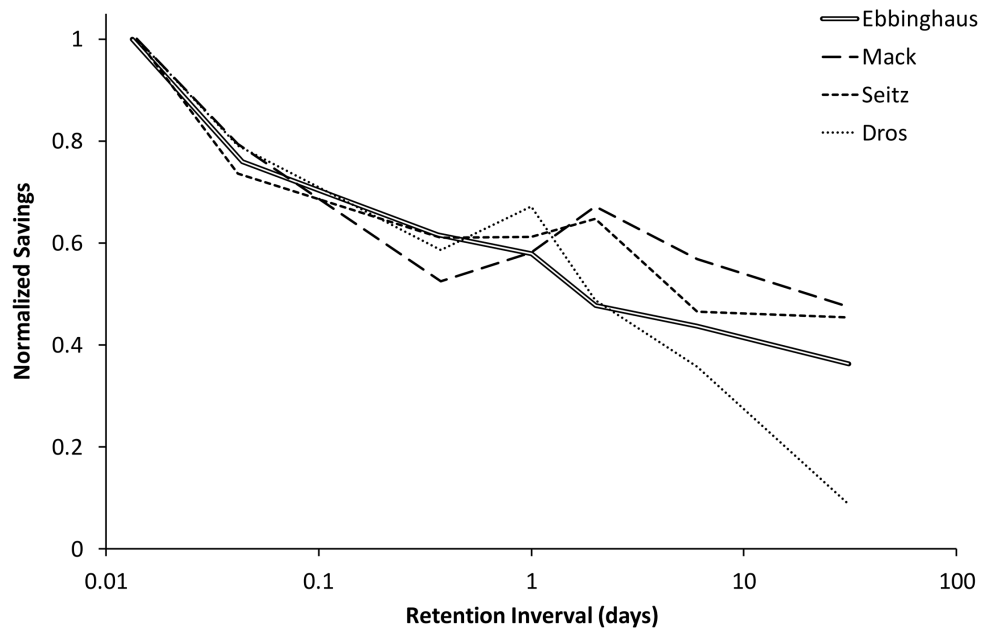


Figure 2.1: Normalized saving scores (the amount of information that was retained) plotted for Ebbinghaus, Mack, Seitz, and Dros[6].

The forgetting curve has important implications for learning and education. It suggests that there is an optimal time for reviewing or reinforcing newly learned material to maximize retention. This is often referred to as the "forgetting window," which is thought to be within the first few days after learning. By reviewing material within this window, it is possible to significantly improve long-term retention and reduce the effects of the forgetting curve.

2.1.3 Spaced repetition

One way to combat the forgetting curve and improve retention is through the use of spaced repetition, a learning technique that involves reviewing and reinforcing material at increasing intervals of time, ideally with active recall. This technique has been widely studied and has been found to be highly effective.

One of the key findings in research on spaced repetition is that it can lead to significantly better retention of information compared to traditional methods of learning. For example, a study conducted by Cepeda, et al.[1] found that spaced repetition was significantly more effective than massed practice (cramming) in terms of long-term retention of information. The authors of this study concluded that spaced repetition is an effective method for improving memory performance. Dunlosky, et al.[2] also found that distributed practice has high utility.

In conclusion, the research on spaced repetition suggests that it is a very effective learning technique that can help individuals improve their recall of new information.

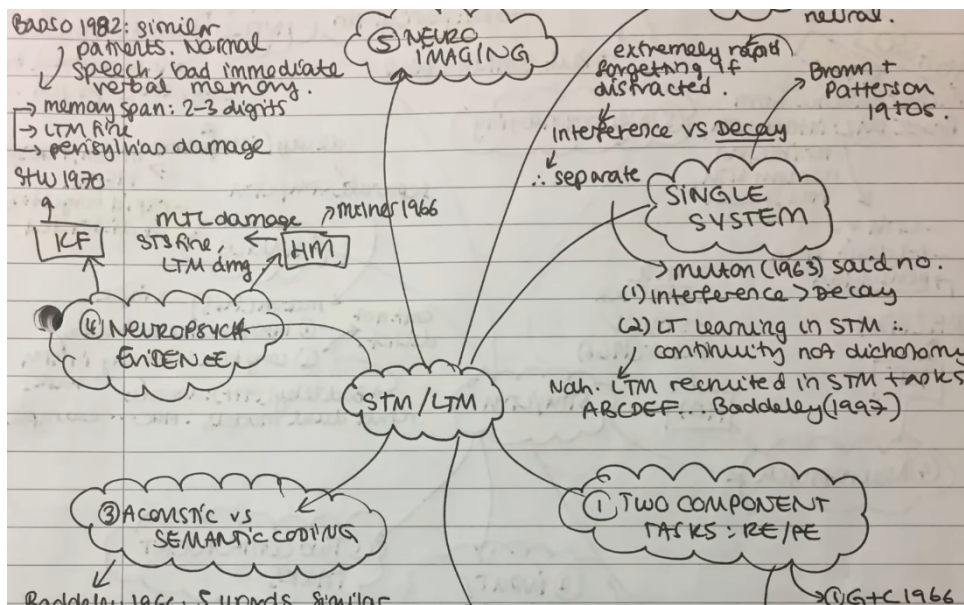


Figure 2.2: Concept map example by Dr. Ali Abdaal.

2.1.4 Concept mapping

Concept mapping is a visual representation of the relationships between concepts. It is often used as a learning and teaching tool because it can help students organize and structure their knowledge. You can see a concept map example in figure 2.2.

Research has shown that concept mapping can be an effective tool for improving students' comprehension, retention, and recall. In a study published in the *Journal of Educational Psychology*, Novak and Gowin[8] found that students who used concept maps to learn scientific concepts performed significantly better on a post-test than students who did not use concept maps. Similarly, this meta-analysis[7] found that concept mapping was an effective instructional strategy for improving student learning across a variety of subject areas and grade levels.

One reason why concept mapping may be effective for learning is that by creating a visual representation of the relationships between concepts, students can see the connections between different pieces of information and understand how they fit together. This can make the information easier to remember and retrieve later.

2.2 Design analysis

Here I explain my reasoning behind some of the overarching design decisions I made for the project and how they compare to other note-taking apps.

■ 2.2.1 Alternative to second brain

A lot of the note-taking apps seem to encourage the idea to create a second brain, where everything you learn will be categorized and saved into neatly formatted notes. I've fallen into the trap myself, when after carefully copy-pasting paragraphs from articles and lectures into dozens and hundreds of notes I couldn't recall most of what I've learned after a few weeks.

Although being able to access everything you've learned at any time may be appealing to some, my idea is to make an app that by design will force the user to take and review notes in a way that will eventually remove the need to use the app altogether because all the information will be stored in the user's long-term memory.

■ 2.2.2 Generic vs Specific use case

Another issue is that most of the apps are one-size-fits-all. The following use cases are very naturally separable: journaling, to-do lists, and learning. Yet a lot of the mainstream note-taking apps have generic designs to fit all these cases. By doing so they fail to target the unique problems of each use case.

The idea of having everything in one place is appealing, but trying to fit every use case at the same time creates friction for the user that only needed to do one thing in the first place. After a while of using such apps, I started to see all the ways it could be improved if it was made just for learning.

■ 2.2.3 Tag system vs File explorer

There are three approaches to organizing notes: flat tag system where each note can be assigned one or multiple tags and a user can see all the notes with a specified tag as can be seen in figure 2.3 (a), hierarchical file explorer structure where notes are the leaves of a tree made of folders, like in figure 2.3 (b), and a mixture of both.

Having both tags and hierarchies, in my opinion, will only create confusion because now each time a user creates a note he has to decide in which folder to store it and then whether to tag it or not and then which tag to give it.

The benefit of a tag system is that it allows a single note to have multiple tags and be seen in multiple contexts. The file explorer solves this problem with shortcuts. The big drawback of tags is their inherently flat structure. For example, creating a tag for each university subject will quickly make a huge mess. So instead users are encouraged to create a few broad categories. This leaves the responsibility of marking (with colour, text, etc) notes related to one subject to the user, which makes it harder to look up notes related to a subject over time.

■ 2.2.4 Spatial file explorer

There are two ways to search for a note: when you can recall keywords from a note and use text search to find it and when you can't and so you have

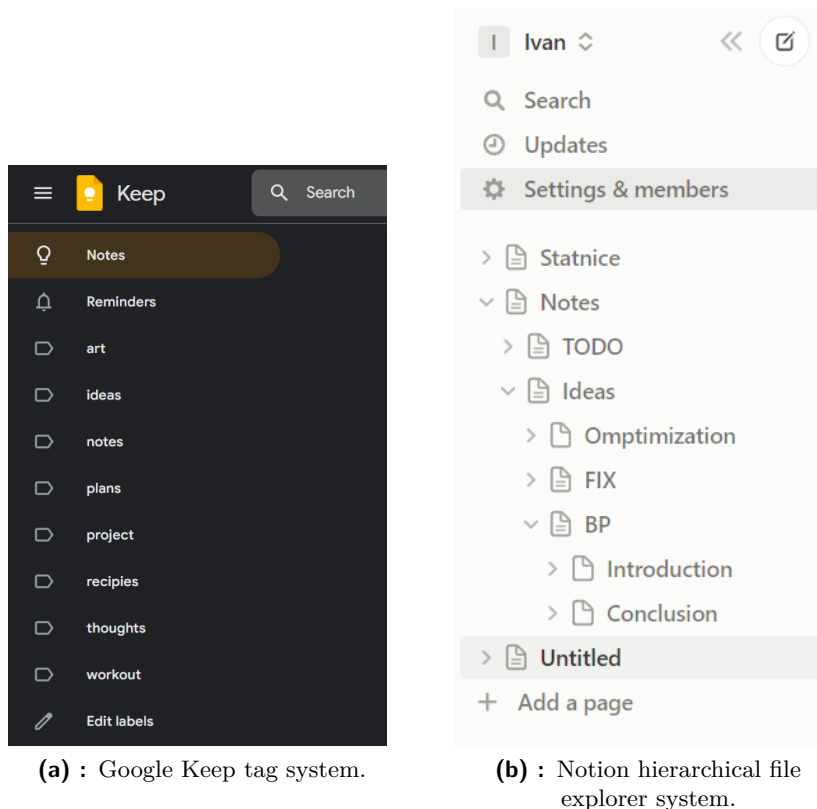


Figure 2.3: Examples of note-taking apps using tags and file explorer

to go through notes with a particular tag in chronological order until you find the one you were looking for. The latter case is frustrating and not that uncommon.

This problem can be largely avoided if you use file explorer. Traversing all the notes in order takes $O(n)$ time, while going through a file hierarchy only takes $O(\log(n))$.

An idea that can make the search even faster is to organize notes not only hierarchically but also spatially (the way mind maps do it). Humans are naturally good at remembering spatial information, i.e. locations of objects relative to one another. For example, imagine playing a real-time strategy game with a top-down view. You know without thinking where your base is and where the enemy base is, and where to get wood, and where the river goes. But can you remember at which index you wrote down "toilet paper" in your shopping list?

One way to take advantage of this ability to effortlessly remember places of objects is to put notes and folders on a 2D grid.

2.2.5 Article vs Card note format

Traditionally when you open a note you expect to see all of its contents immediately. For example, in figure 2.4 (a) Google Keep shows you the the

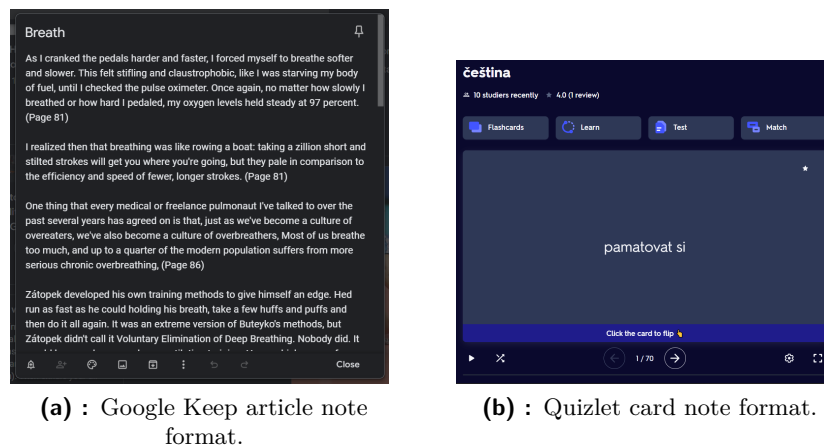


Figure 2.4: Visual comparison of article and card note format

entire note content. This implicitly encourages rereading the information you may have forgotten which goes directly against the Active Recall approach: trying to recall as much as possible before checking the answer.

The solution is to store note contents in a list of cards, like in figure 2.4 (b). One side of a card would contain a question, and another - the answer to that question. On opening a note the user will only see a question and will be expected to give an answer before checking the actual answer.

2.2.6 Rigid vs Free spaced repetition schedule

There are two approaches to implementing spaced repetition. The first one is when you create a rigid schedule of review times based on the forgetting curve for each subject you're studying. The problem with this approach is that a) it doesn't adapt review times to the difficulty of the topic because the decision of review times was made before the actual reviews started and b) you don't know the future and so the system easily breaks when something unexpected inevitably comes up and prevents you from doing a review.

The second approach basically involves delaying the decision of what to review next right to the point when you're about to review it. When the user sits down to study, he is presented with a set of notes that need to be reviewed. It's then up to him to prioritize which notes to review and which ones to leave for the next time. This way the user is in control of when the reviews happen and the system can adapt to different difficulties of each topic.

2.2.7 Private vs Public notes

The idea that your notes are private by default and only you can see them is taken for granted in all of the note-taking apps. You can share a note with people by explicitly inviting users. It makes sense to do that if the notes can have anything from journal entries to birth dates. If, however, we consider

that the notes are used only for studying then the privacy assumption quickly falls apart.

Making notes public by default could encourage users to explore what others are studying, see how they organize their workspace and learn how to better formulate questions.

Moreover, because the notes do not have the traditional structure of a wiki article, but are presented as a set of questions, it could spark curiosity in the reader to find his own answers before flipping cards.

Chapter 3

Implementation

3.1 3-layer architecture

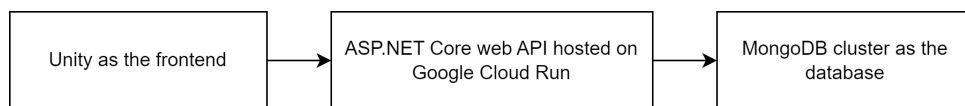


Figure 3.1: 3-layer web app architecture.

Figure 3.1 shows the application divided into 3 layers: a presentation layer (frontend), in this case, it's a Unity project build, an application layer (backend), which is a .NET web API, and, finally, a data layer – a MongoDB cluster.

3.2 Tools

The Unity project as well as .NET web API are written in the C Sharp programming language. OpenAPI specification was used to generate a client library that the Unity project uses to call the web API. The web API is hosted on Google Cloud Run and is connected to a Database-as-a-Service MongoDB cluster.

3.2.1 C Sharp

C Sharp is a general-purpose, object-oriented programming language developed by Microsoft. It was designed to be simple, modern, and type-safe, with features that promote robust software development such as automatic garbage collection and strong type-checking.

3.2.2 Unity

Unity is a cross-platform game engine that is widely used in the industry to build a variety of games. Unity is known for its powerful visual editor, scripting tools, and a range of built-in components. It uses C Sharp as its primary programming language.

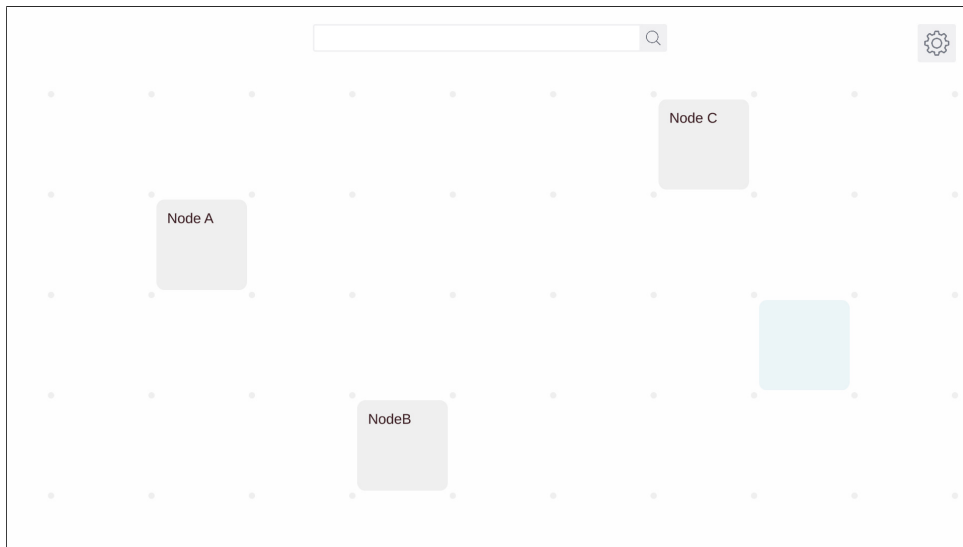


Figure 3.2: Desk example.

1. Collect all user input such as key presses and mouse movement and process it to detect things like double clicks.
2. Move the camera or change its size.
3. Get a reference to an item under the user's mouse.
4. Based on the state of the desk, the user's input and the item, perform one of the following:
 - Open nodes or folders
 - Show links to and from shortcuts
 - Select, deselect, and delete items
 - Drag items
 - Rename folders
 - Rescale folders
 - Create nodes, folders or links

The steps where the code handles dragging and selection use a property of OOP called polymorphism. Polymorphism allows objects of different types to be treated as if they were of the same type because they inherit from the same class.

■ 3.3.2 DeskItem, Node and Folder classes

Both Folder and Node classes inherit from DeskItem class, which has defined public methods, virtual properties, protected virtual methods, public virtual methods, and even abstract methods. I have tried many ways to simplify this mess, but the final solution, although complex, is the best I've found.

on the following line. Note that, after the user entered a name, there is a callback that opens a folder if it's large enough relative to the screen. Finally, the desks' bounding rectangle is updated.

3.3.3 Shortcuts

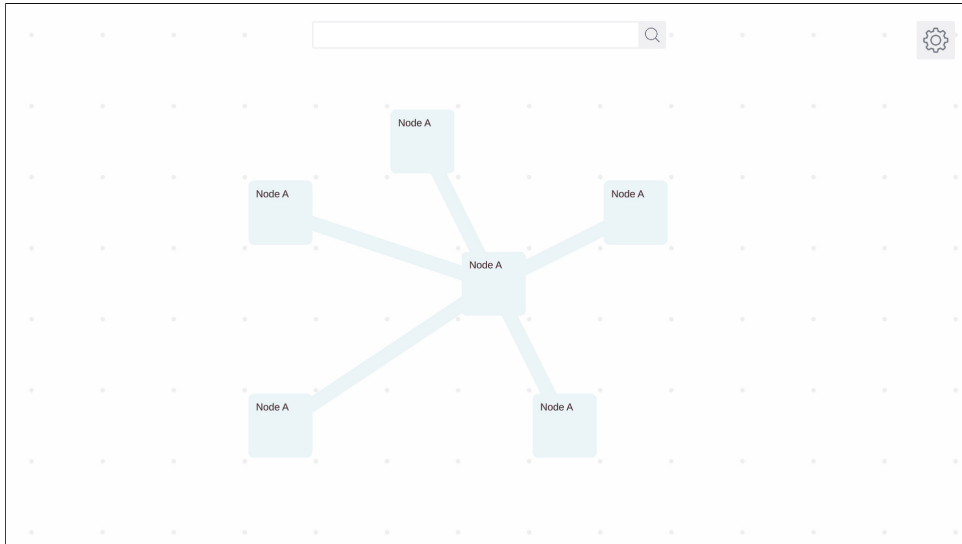


Figure 3.3: Shortcuts.

If you press and drag the right mouse button on a node you will create a shortcut to that node (figure 3.3). In code and in the database it has the same schema as a node. The difference is that its `LinkedFrom` field stores the id of its origin. And its `NoteId` field always points to the same note as the origin. And if you delete the origin, all of the shortcuts will be deleted as well.

Their purpose is to let you cross-reference the material so that you can see links to the same material in multiple contexts. For example, you can create a shortcut of your "Matrix Multiplication" node from the "Linear Algebra" folder in "Graphics Programming" or in "Optimization".

3.3.4 Level of Detail

When a user zooms in, an event is raised, to which every folder is subscribed. If a folder is sufficiently zoomed in, then it opens and all of its contents are drawn (figure 3.4). When a folder takes less than a specified percentage of screen height or width then it closes and gets filled with solid color (figure 3.5). This improves performance, but it also reduces the user's cognitive load. Furthermore, the folder can only be moved or renamed when it is in a closed state.

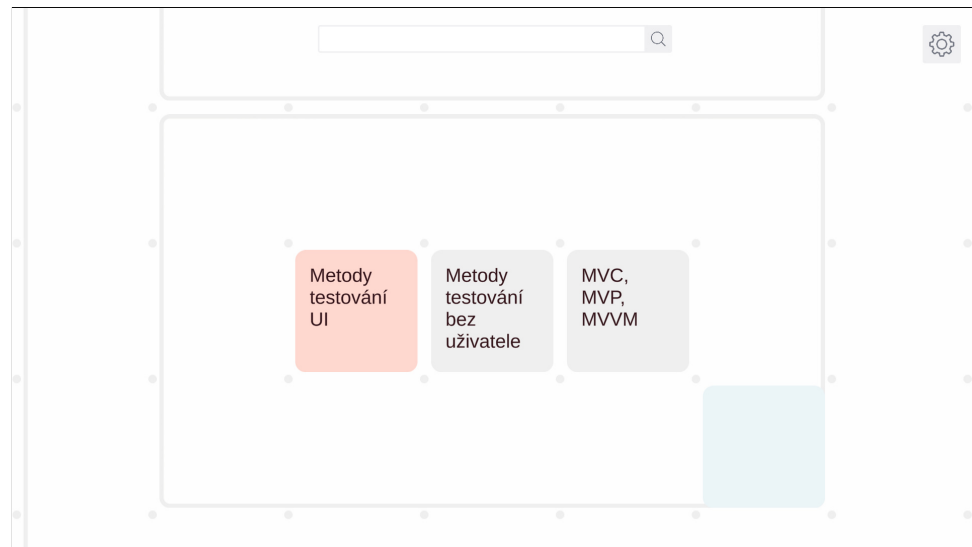


Figure 3.4: Open folder.

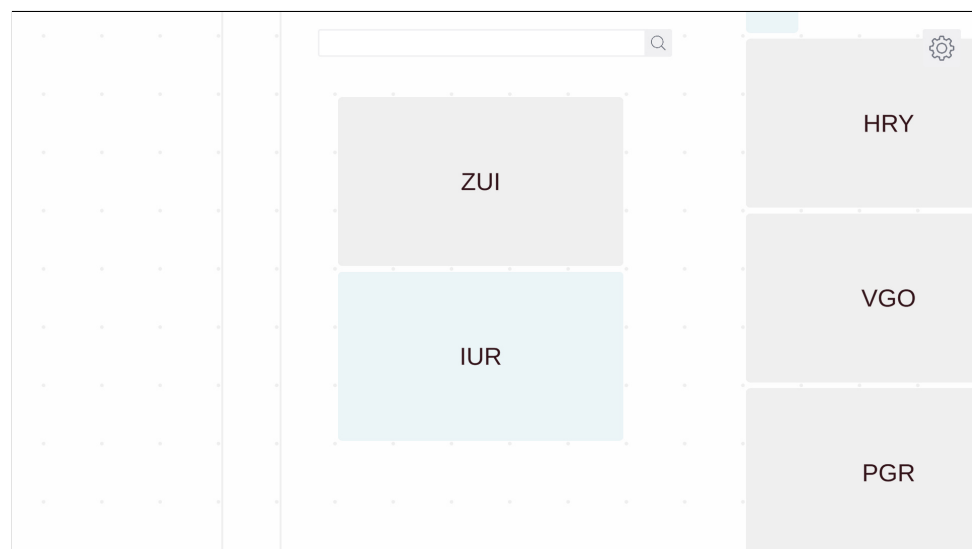


Figure 3.5: Closed folder.

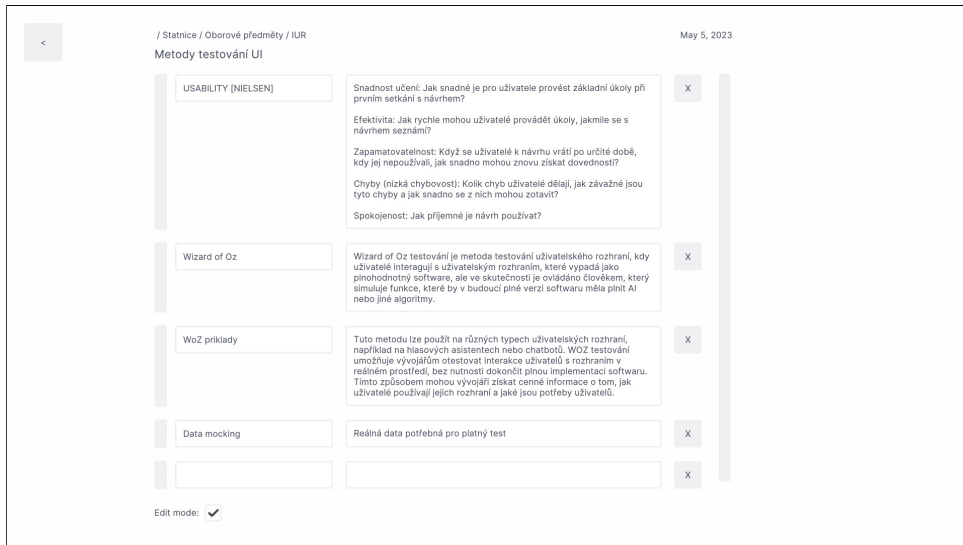


Figure 3.6: Edit mode.

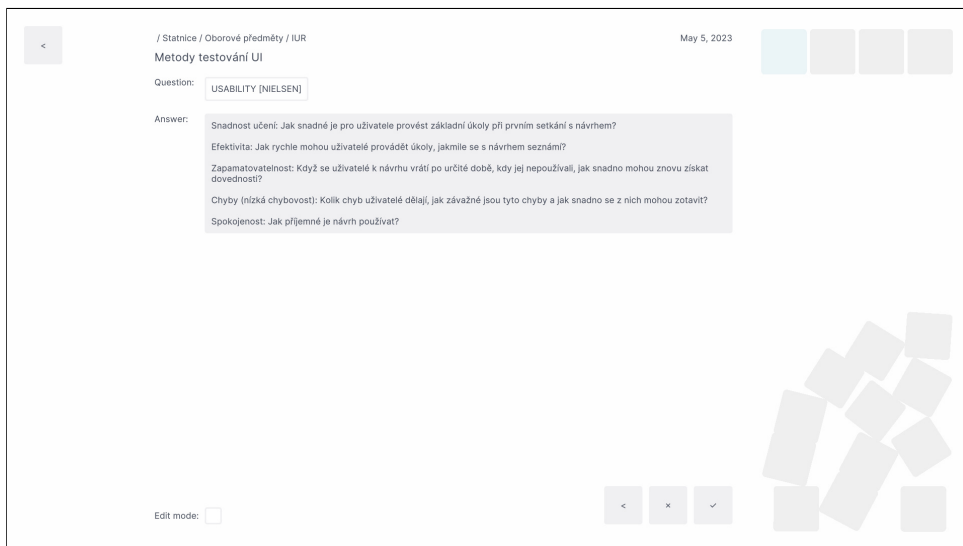


Figure 3.7: Review mode.

mostly incompatible. Initially, I started with uGUI since I was familiar with it and it is generally more widely used in the community (figure 3.8).

The biggest problem was that making the UI behave any differently from the predefined example produced lots of unexpected behaviour. It is very rigid around its intended use case, so making anything custom will require you to either modify the source code or write obscure classes on top of it that mute and/or add functionality to them.

For example, for folder titles, I used uGUI since UI Toolkit is not supported in world space. I wanted the titles to be multiline. When I tested it, it worked fine, except if I hit enter when I finished typing a title, it inserted a new line before unfocusing causing the whole text to visually slide up the folder body. There is no toggle in the text field component that disables this. I had to make a copy of the text field component, find the part responsible for inserting new lines in 3000+ lines of code and modify it without breaking anything to make this simple change. This was only the tip of the iceberg.

It was very frustrating to get even the simplest things working consistently when I was implementing the note view in uGUI so I eventually decided to switch over to UI Toolkit. The shortest way to explain UI Toolkit is that it's basically XAML or HTML with CSS but with bugs and less functionality. Still, it was much better than uGUI and I was able to build the note interface I wanted.

In conclusion, this is the weakest part of the engine and by extension the worst UI system I have ever worked with.

■ 3.3.6 GameModeManager class

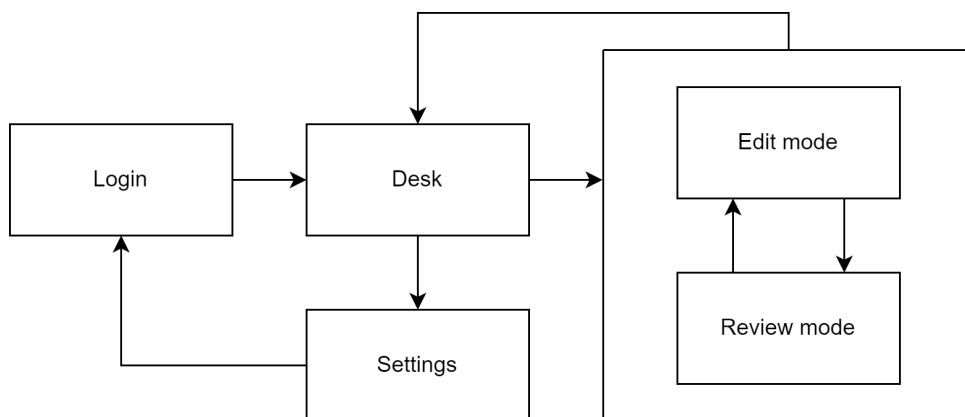


Figure 3.9: Finite State Machine at the core of the Unity project

Finite State Machines (FSM) are a popular way to structure code in game development. Firstly, they reduce the growth of nested conditional statements as the number of states in an application increases. Secondly, they let the states manage their own entry and exit logic, which has comparable benefits to OOP.

In this application, they're primarily used for the second reason. Transitions handle state cleanup and persistent storage. There are two FSMs in total: the one in GameModemanger class that handles the transition between the login page, desk view, note view, and settings, and the FSM inside the note view (figure 3.9).

The code responsible for state transitions is very simple:

```
public static void TransitionFromTo(ref State from, State to)
{
    from?.OnEnd?.Invoke();
    from = to;
    from?.OnStart?.Invoke();
}
```

■ 3.3.7 SaveManager class

The web API generated client code is not used directly. Instead, there is a level of abstraction in the form of SaveManager class. It neatly packages all of the endpoints I use in one class and also stores a reference to the currently loaded desk and the user's desk. Doing that lets it gatekeep any write calls to a visited desk and of course lets a user go back to their desk.

As an example, here's a function UpdateNote:

```
public void UpdateNote(Note note)
{
    if (!myDesk) return;
    NoteDTO noteDto = new NoteDTO(note.Cards, note.Reviews);
    desksApi.DesksDeskIdNotesIdPut(userDesk.Id, note.Id, noteDto);
}
```

■ 3.3.8 ZoomController class

The ZoomController class is responsible for animating camera position and size and keeping track of the desk bounding rectangle. The animation is done using tweening. The tweening implementation is provided by the LeanTween package.

Tweening is a technique used in computer graphics and animation to create smooth transitions between two states of an object over time. It involves defining the starting and ending states of an object and then calculating intermediate states at regular time intervals to create an animation.

The two use cases of this class are searching and reframing the desk contents. The position and size returned from the search endpoint are passed directly to the AnimateSearch function. And if you type in the search bar "I'm lost" and hit Enter, the ZoomOut function will position the camera so that all of the desk contents are visible using the stored bounding box.

3.3.9 Object pools

Object pools work by allocating a set number of objects in memory and then reusing them as needed. When an object is no longer needed, it is returned to the pool instead of being destroyed. This reduces the overhead of allocating memory of new objects and performance drops after destroying objects caused by the C Sharp garbage collector.

In the case of the app, object pools were used for the desk elements to improve performance when switching between desks. When the user switches between desks, the app simply reuses the desk elements that are already in memory instead of creating new ones. This removes the need to allocate and deallocate potentially hundreds of game objects when a user switches desks.

A big issue I faced with object pools is properly cleaning up an object's state when it is released back into a pool. Things like titles, sprite sheet, children, parent, links, and selection. One of the most difficult bugs to find was due to an event that was being raised on a released object. The event is a part of the LOD system that can enable or disable game objects based on camera size. The problem was that releasing an object to the pool is in reality just deactivating it and so when the event is raised to activate it you get a zombie object that a user can interact with but that has no state.

3.3.10 Search bar

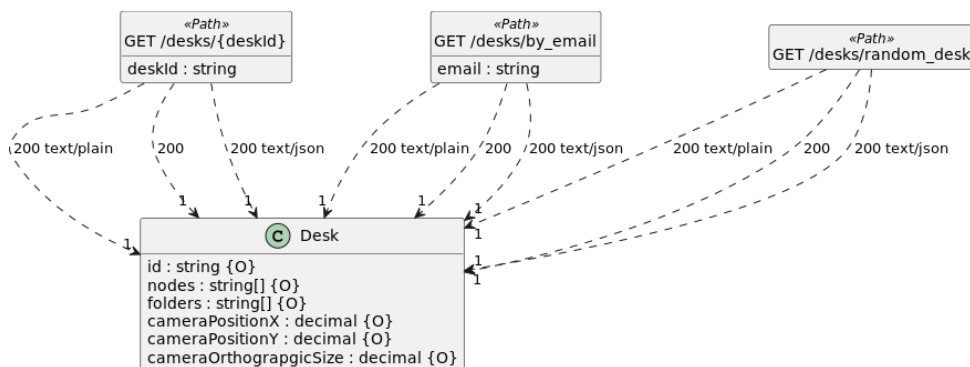


Figure 3.10: Desk GET endpoints.

The search bar has multiple purposes. When you enter something into a search bar and press enter it cascades through a series of if statements to decide what exactly do you want. Here's how it looks in code:

```

public void Search(string searchTerm)
{
    if (string.Equals(searchTerm, "i'm feeling lucky"))
    {
        Desk desk = SaveManager.Instance.GetRandomDesk();
        SwitchDesks(desk);
    }
}
  
```

```
else if (string.Equals(searchTerm, "i'm lost"))
{
    ZoomController.Instance.ZoomOut();
}
else if (IsEmail(searchTerm))
{
    if (searchTerm == PlayerPrefs.GetString("email"))
    {
        return;
    }
    Desk desk = SaveManager.Instance.GetByEmail(searchTerm);
    SwitchDesks(desk);
}
else
{
    var sr = SaveManager.Instance.Search(searchTerm);
    ZoomController.Instance.AnimateSearch(sr.X, sr.Y,
                                        sr.SizeX, sr.SizeY);
}
}
```

The "I'm feeling lucky" was inspired by Google main search page and loads a random desk from the collection of all desks. This was added to encourage exploration even if a user doesn't know specific emails.

The "I'm lost" was added after testing, when users accidentally dragged the camera too far from where their folders were. It calls the `ZoomController` to reframe the contents of the desk.

When you enter an email, a desk that belongs to it will be loaded as shown in figure 3.10, provided it exists. You can open any node or folder you like but the changes you make are not persisted.

Finally, if it's none of the above, the search term you entered will be used in MongoDB full-text search engine to find a node or a folder that has a matching title. The camera will then zoom in on the search result.

■ 3.3.11 Logging

In the final stages of development, and especially during testing, I had to debug builds. Sometimes it was obvious where the problem was, but most times it wasn't. The solution was to add logging to every major part of the project. I then intercept these logs and write them to a file in the "Logs" folder. Each run creates a new log file. It's then pretty easy to recreate the user's actions just by following the logs, but usually, it wasn't necessary since one error message was enough to find the bug.

3.4 Backend

The backend consists of a Google Cloud Run hosted .NET web API that is connected to a remote MongoDB cluster. It handles persistence, authentication and authorization, search, and access to other desks.

3.4.1 Layered backend architecture

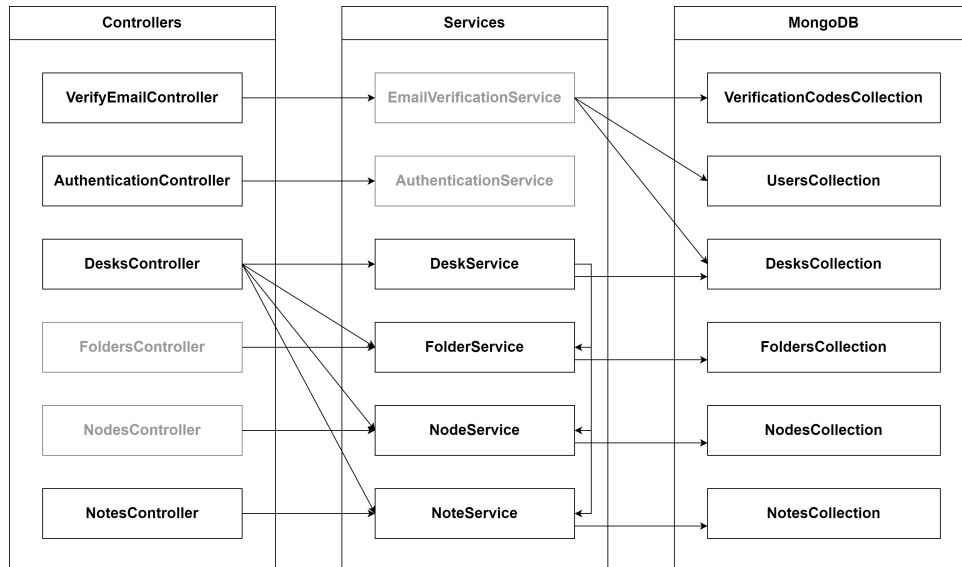


Figure 3.11: Layered backend architecture.

This was my first experience designing an API, so I added functionality to the API as I learned how to build one. The resulting design is therefore messy and evolutionary-looking.

For example, you can see in figure 3.11 that there are endpoints for Desks, Folders, and Nodes, which I created at the very start of development. As I then learned about how resources are structured in REST APIs most of that functionality went into the Desks endpoint, while the Folders and Nodes endpoints were left unused. They're still available, but require an Admin role because they don't require the caller to be the owner of the desk to write to it.

Another problem is that a lot of the functionality that is in controllers should be moved to services. The extreme cases of this the `AuthenticationService` and `EmailVerificationService` which have no functionality and just sit there in a vacuum waiting for the codebase to be refactored.

Designing the bottom-up way is faster but produces messier results, but in cases where you don't know what you're building at the start, which was certainly my case, it is the only way. And, in my opinion, it's a better way because you usually don't know a good design for your application, unless you've tried 10 bad ones.

3.4.2 Session vs Token Authentication

There are two main approaches to user authentication: sessions and tokens. Sessions involve a stateful session between the front-end client and the back-end server, where a session ID is created and stored in the database and the client. Each request must then do a database lookup to check the session id validity, which can cause bottlenecks. Token-based authentication, on the other hand, generates a JSON web token (JWT) that doesn't need a database lookup to validate. The way it works is as follows: the token is created with a private key on the server, sent back to the client, and then on each following request the server only needs to validate the signature.

3.4.3 Registration flow

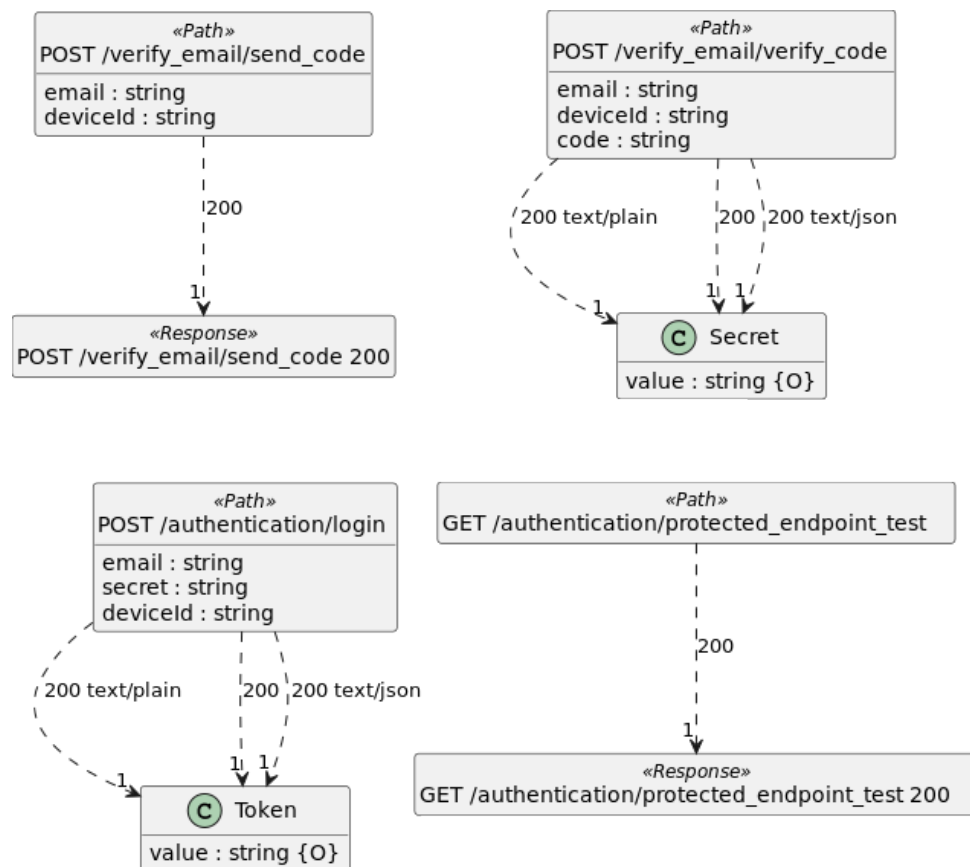


Figure 3.12: Endpoints responsible for authentication and authorization.

This application uses token-based authentication. First, a verification code is sent to the provided email. Once the email is verified, the backend creates a user account and sends the secret (password) to the client (figure 3.12). Note that one user can have many devices linked to his account and each account will have a different secret. A correct combination of user email, device id, and a secret will make the server return a token. This token is then attached

to the HTTP authorization header prefixed by “Bearer”. Afterwards, the token can be tested by making a GET /authentication/protected endpoint test request. It will return the user’s email, desk, device id, and role. The majority of endpoints can only be accessed by authorized users.

■ 3.4.4 Read/write permission

Each endpoint that reads data from a desk requires the user to be authorized. Each endpoint that writes data to a desk requires that the user is authorized and the desk id stored in the JWT matches the desk id that the user is trying to write to. This works because if you were to modify the desk id in the JWT, the signature will no longer be valid and the API won’t let you through.

■ 3.4.5 Data Transfer Objects

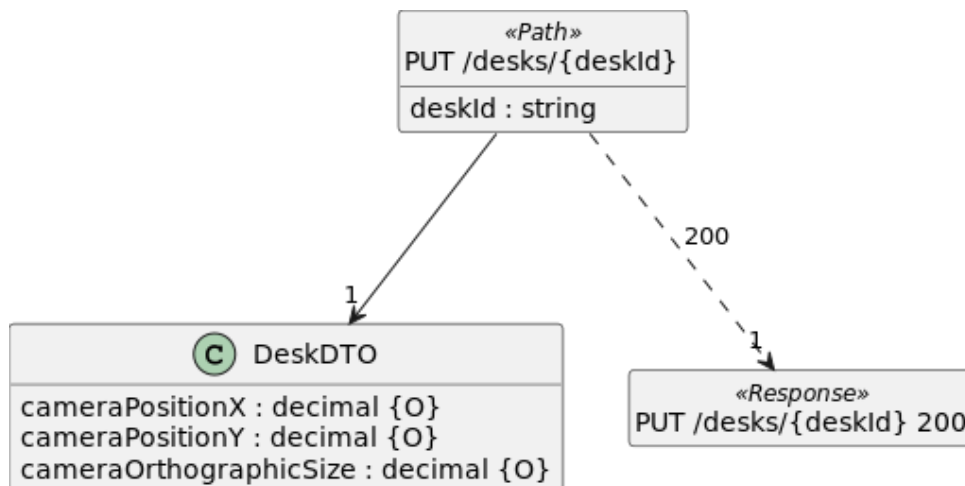


Figure 3.13: Endpoint using DeskDTO for updates

Most create and update requests use Data Transfer Objects (DTOs) to prevent API users from thinking they could update data they shouldn’t. If a Desk class has id, nodes, folders, and camera position fields then the DeskDTO class only has camera position (figure 3.13). This way it’s clear to the user which fields he can/ should modify.

■ 3.4.6 Rate limiting

The API has a simple rate-limiter enabled, which limits the number of requests per IP per minute to a 100.

■ 3.4.7 Search implementation

The search works by doing full-text search queries in MongoDB collections. First, the algorithm searches the titles of folders. Second, if nothing was

found, it searches the titles of nodes. Finally, the position and size of the most relevant result are returned.

■ 3.4.8 Review priority algorithm

One popular method of implementing spaced repetition is the Leitner system, which uses a series of virtual "boxes" to store flashcards, with each box corresponding to a different level of difficulty. Initially, all of the flashcards are placed in the first box. When a student studies a flashcard and is able to recall the information on it, the flashcard is moved to the next box. If the student is unable to recall the information, the flashcard is left in the current box. Over time, the flashcards that are more difficult to remember will be presented more frequently, while the easier ones will be shown less often. This helps to reinforce the student's memory of the more challenging material.

The review priority algorithm used by this application is a variation of the Leitner system. It outputs either HIGH or LOW and based on that a node is either color grey or red.

First, it initializes the 5 time intervals or "boxes":

1. [0, 1) days
2. [1, 7) days
3. [7, 16) days
4. [16, 35) days
5. [35, 365) days

Then, it sets the maximum reached interval to 0 and goes through every past review. The interval of the current box is found and if it's higher than the current max, it's updated.

Then the algorithm finds the box in which the present time falls and if this box is higher than the max box, it returns HIGH. Otherwise, LOW is returned.

The algorithm is very simplistic and rigid and doesn't do much to adapt to the user's performance or review schedule. However, it's sufficient for demonstrating how the spaced repetition would look in the context of this application.

Chapter 4

Tests and feedback

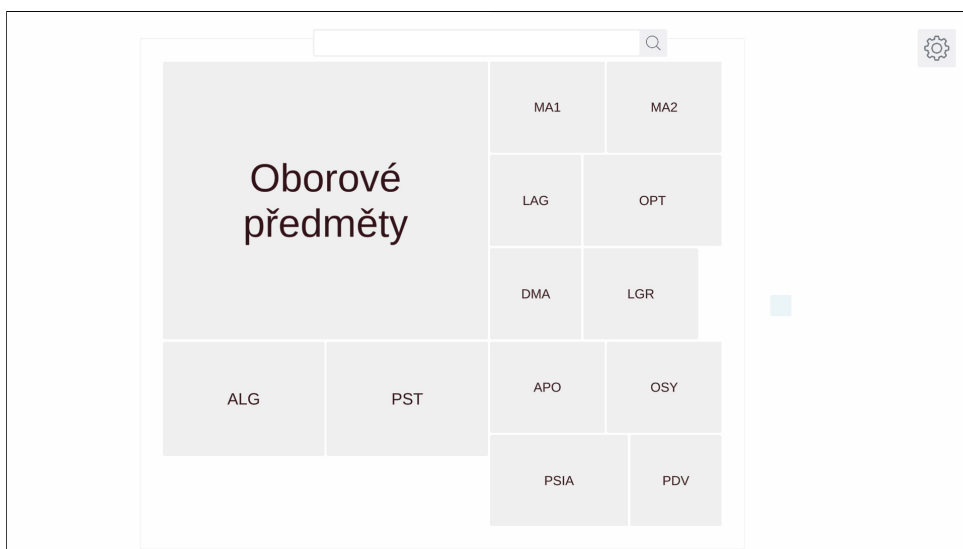


Figure 4.1: Example of a desk filled with subjects.

Testing is crucial for developing any software because it identifies bugs and usability issues. The app was first tested myself (figure 4.1), then in-person, which involved observing users as they interacted with the app and gathering their feedback. Afterwards, bugs were fixed and some of the suggested features were implemented.

4.1 Observations during testing

I tested the app formally on the 4 people that are listed in the section below. Before that, when the app was very unstable, I also did tests informally on around 3 people.

The most common problem I saw when people interacted with the app is unfamiliarity with the interface. For example, the full control of the camera is not at all obvious to the users even if I say that it's there. Only after I showed how the camera can be moved and zoomed did they get it. This is just one example, but there were many. The solution I came up with was to create a

short user manual with actions mapped to keys. And, more importantly, a video tutorial that shows how to interact with the app as well as introduces the viewer to the main concepts behind it.

The complaint I heard most often was the small button size, which I suspect was due to me working on the app in a small editor window in Unity where buttons appeared relatively large. I also wanted the design to feel minimal so in some places I omitted icons, which during testing only caused confusion. So buttons were made bigger and icons were added to indicate their purpose. In addition, I made physics less distracting.

Besides this, I found dozens of bugs, especially in the early informal stages of testing. I added checks for a valid email address and 6-digit code, I found that the review priority of a node wasn't actually updated, I fixed bounding box calculations, improperly cleared state in object pools, incorrect behaviour during card rewind and many, many more that didn't make it into commit messages of the project repository.

4.2 Suggestions received

1. User A, male, VSE, FIR, 2nd year
 - Color code folders
 - Give notes and folders different visual styles because they're easy to mix up
 - Zooming in on a note should open it
 - Make buttons more visible
 - Guide users on the first launch
 - Physics during a review can be distracting
 - Make UI less scattered
 - Add web and mobile version
2. User B, male, CVUT, FIT, 1st year
 - Bigger, more colorful buttons
 - Enforce unique titles for search
 - Preview text for title, question, answer text fields
 - Button icons
 - Ability to switch between notes in note view
 - In review mode, sort cards by difficulty
 - Highlight physics boxes based on whether the answer was correct or not
3. User C, male, CVUT, FEL, 3rd year

- It is not intuitive that after completing a review, the app automatically throws cubes and waits. It would be better to add a button to exit or an automatic exit after a certain amount of time
- It would be better to mark correct and incorrect cubes with green and red colors
- It is not clear why the first cube is blue and why they fall from the end. It would be better if the highlighted cube fell first
- Simulate and show only the physics blocks relevant to the current review session instead of the blocks corresponding to all reviews from the past
- When clicking on the search button with an invalid email, nothing happens. Have a small message pop up that lets the user know that it is not a problem with their internet connection

4. User D, female, future CVUT, FA student

- Make the app windowed because on some devices the window doesn't fold
- Increase the scroll speed in note view or let the user change it in settings
- Bug: at higher list item counts, text fields sometimes deselect on their own
- Mobile version needed very much

4.2.1 Feedback discussion

First, I heard pretty often that I should add folder colouring, for example from User A. I think this is a great idea because it supports the foundational idea behind this app that humans are better able to process visual information. Colour would add another dimension to organizing your notes and will make the whole application more vibrant. Though, I do think it's important to limit the palette to 3-4 colours.

An interesting idea by User B was to present the cards to the user in order of their difficulty and shuffle them a little. I see potential in this idea because I can randomness to this step. Adding randomness to each should increase the challenge and promote understanding instead of memorization. One way to do that is to randomly shuffle cards inside a note for each review. This interleaving of the material has been shown to be more effective for concept learning than studying in blocks[4]. Interleaving can also be added when in the note review sequence. User B suggested adding a way to review the next note without exiting to the desk and this is a great place to inject some randomness. Another thing User B proposed was to add the ability to go through all the search results in order of relevance.

My personal suggestion is to add keyboard shortcuts. Although intuitive, interacting mainly with a mouse can be slower compared to a keyboard.

Adding shortcuts for all the actions in the app can increase productivity for many users willing to learn them. Also, having commands that a user can input in a search bar for actions like creating, or moving a desk element can make the user experience even smoother.

Chapter 5

Discussion and limitations

5.1 Lacking features

Here are some potential development ideas that didn't make it into the project due to finite time.

Undo and redo functionality can be implemented using event sourcing, where every change is recorded as an event and can be used to revert the data to a previous state. This will also let a user see a timeline of their desk.

Image support inside a note is also an important feature for many people because a lot of notes are taken with screenshots of lecture slides, diagrams, or photos.

Offline support will allow users who don't always have internet access to use the app. This feature, however, combined with multiple device support will require an automatic and stable merging algorithm.

Another useful feature would be to make folders “rubber” so that they rescale to fit their content. If you, for example, drag a note onto the folder edge, it will expand there. If you delete a node inside, the folder will scale down to the new bounding box of its children.

There are several useful ideas from repository hosting services that can benefit this project:

- When viewing other desks, it should be possible to fork any node or folder to your own desk.
- Each desk has a one-to-one mapping to its owner, and the owner is the only one who can write to a desk. A useful feature would be to allow the desk creator to give multiple users edit rights to allow collaboration.
- For users who want their notes to be private, there should be a toggle that would hide their desks from the public.

5.2 Optimization

Optimization is not currently an issue, but the application was only ever tested with node and folder counts up to a few hundred. I expect it will

crumble when that count reaches a few thousand or tens of thousands of desk items. And although these numbers seem large for a typical use case, over something like 5 years, I can definitely take a thousand notes. I see a few ways to speed up the app:

- Make API calls async.
- Batch the update calls: instead of calling the API update endpoint on each child after a folder is moved, batch all requests into one, or better yet, update the children without the need for any more calls.
- Only open the folders that are within camera view; to do that use some form of spatial partitioning algorithms like uniform grids or quadtrees.
- Experiment with Unity Jobs and Burst compiler to utilize parallelism and apply compiler optimizations.

■ 5.3 AI applications

There are several ways to incorporate recent advances in machine learning and large language models (LLM) into the project to improve it.

■ 5.3.1 Trainable spaced repetition algorithm

The spaced repetition algorithm can be replaced with Duolingo's HLR model[9]. This algorithm adapts to the learner's ability and the difficulty of the material using a custom regression model. Having the predicted forgetting rate would also allow filling node icons like progress bars, indicating the probability of forgetting a node.

■ 5.3.2 Semantic search

At some point I want to implement a semantic search, that is, search not by an exact match of symbols, but rather by meaning. To do this all the searchable content needs to be preprocessed using an embeddings API, embeddings need to be generated for each note, and then compared to determine their similarity. Finally, the search can be performed by encoding the query as an embedding and finding the notes with the closest embeddings.

■ 5.3.3 Autocomplete questions


There is no actual need for the user to write questions for their notes since a question can be inferred from the answer. It is possible to connect the app to an LLM API and generate questions automatically. In addition, questions need not be the same for each review. Adding variation and randomness to the questions would promote deeper understanding because the user wouldn't be able to just memorize a mapping of questions to answers. This way a user will have to look at the material from a slightly different angle each review.

■ 5.3.4 Check answers with AI

A step further would be to use the same API to input an answer for a generated question and ask if it was correct. This way a student wouldn't be tempted to just reread a note or mark his answer as correct even if it was incomplete. Typing out an answer is of course slower, but it requires more effort, which we know from active recall research makes our brain more likely to remember the information.

■ 5.3.5 Recommendation system

Exploring other desks is random in the current version, but it doesn't have to be this way. Implementing a recommendation system based on likes and/or note content can let users find others with shared interests. This can be done by comparing every node from every desk to each other the way it would be done with semantic search within one desk. Given a node, once we have its matching node in another desk, we can insert its copy next to the original, give it a different colour, and link it to its parent desk. This is just one approach however, more research into recommendation systems is required.



Chapter 6

Conclusion

In this thesis, I summarized research that supports active recall, spaced repetition, and concept maps. I then analyzed and challenged different approaches and assumptions that come up when designing a note-taking app. I explained why the notes need to be in a card format, why there is no harm in making notes public and why the notes are located on a 2d plane.

In the "Implementation" chapter I went over the technologies I used for the project, like C Sharp, Unity, and MongoDB. I took a deep dive into the overall architecture, the structure of frontend and backend projects, and the logic flow of individual scripts and functions. I showed how the level of detail, persistence, search, spaced repetition algorithm, authorization and authentication were implemented.

Afterwards, I tested the application on several users, found and fixed bugs, gathered feedback and made relevant changes. This resulted in a concise user manual, a tutorial video, and UI that's more readable and easier to navigate. I then discussed other promising suggestions that were outside the scope of the project, like adding offline support and allowing images inside notes, and possible integrations with AI.

So, in this one note-taking app built with a game engine I combined effective learning strategies and made it so everyone's notes are freely accessible. I tested it on real users and wrote about the trajectory this project might take in the future.



Bibliography

- [1] Nicholas J Cepeda, Harold Pashler, Edward Vul, John T Wixted, and Doug Rohrer. Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychol Bull*, 132(3):354–380, May 2006.
- [2] John Dunlosky, Katherine A. Rawson, Elizabeth J. Marsh, Mitchell J. Nathan, and Daniel T. Willingham. Improving students’ learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest*, 14(1):4–58, 2013. PMID: 26173288.
- [3] Hermann Ebbinghaus. Memory: a contribution to experimental psychology. *Ann Neurosci*, 20(4):155–156, October 2013.
- [4] Jonathan Firth, Ian Rivers, and James Boyle. A systematic review of interleaving as a concept learning strategy. *Review of Education*, 9, 03 2021.
- [5] Jeffrey D. Karpicke and Janell R. Blunt. Retrieval practice produces more learning than elaborative studying with concept mapping. *Science*, 331(6018):772–775, 2011.
- [6] Jaap M. J. Murre and Joeri Dros. Replication and analysis of ebbinghaus’ forgetting curve. *PLOS ONE*, 10(7):1–23, 07 2015.
- [7] John Nesbit and Olusola Adesope. Learning with concept and knowledge maps: A meta-analysis. *Review of Educational Research*, 76:413–448, 09 2006.
- [8] Joseph Novak and Alberto Cañas. The theory underlying concept maps and how to construct them. 01 2006.
- [9] Burr Settles and Brendan Meeder. A trainable spaced repetition model for language learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1848–1858, Berlin, Germany, August 2016. Association for Computational Linguistics.



Appendix A

Contents of electronic appendix

- bin/: the Unity project Windows build
- src/: the source code files of the project.
- latex/: LaTeX files
- images/: screenshots of the project.
- WATCHME: video introduction to the application.
- README: key bindings and additional information.

Appendix B

User manual

B.1 Desk view

- Drag camera = mousewheel hold + mouse drag
- Zoom in and out = scroll mousewheel
- Create and open a note = RMB click on empty
- Create shortcut = RMB hold on note + mouse drag
- Create folder = RMB hold on empty space + mouse drag
- Rename folder = RMB click on closed folder
- Open item = LMB double click on item
- Drag item = LMB hold on item + mouse drag
- Select item = click LMB on item
- Unselect all = click LMB on empty space
- Delete selected items = press Delete
- Search item by its title and zoom in on it = search "title"
- Frame all items = search "i'm lost"
- Load random desk = search "i'm feeling lucky"
- Load desk by email = search "email@gmail.com"
- Go back to your desk = press button at top-left
- Open settings = press button at top-right
- Exit = open settings and press "Exit" button
- Logout = open settings and press "Logout" button
- Close settings = press button at top-right

Red note highlight means its time for a review.

■ B.2 Note view

- Close note = press button at top-left
- Toggle edit/review mode = press toggle at bottom-left

■ B.2.1 Note view edit mode

- New list items are added automatically once the last list item is used
- Items can be deleted
- Items can be reordered
- Scroll bar appears once items fill up the screen

■ B.2.2 Note view review mode

- Answer is displayed on mouse hover
- Check mark button rates your answer as correct and shows next card
- Cross button rates your answer as incorrect and shows next card
- Left arrow button shows previous card