**The Sliding puzzle project:   
The Trickling Enigma**

Contents

[1. Idea origin 2](#_Toc106115251)

[2. The Game 2](#_Toc106115252)

[3. Service-Oriented Architecture 3](#_Toc106115253)

[3.1 Model 4](#_Toc106115254)

[4. Programming languages 5](#_Toc106115255)

[4.1 Angular 5](#_Toc106115256)

[4.1.1 PIXI.js 5](#_Toc106115257)

[4.2 Java SpringBoot 5](#_Toc106115258)

[4.3 SQL 6](#_Toc106115259)

[5.Requirements 6](#_Toc106115260)

[5.1 Functional Requirements + MoSCoW 6](#_Toc106115261)

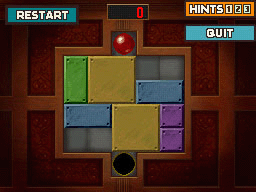
[5.2 Non-Functional Requirements 7](#_Toc106115262)

[6.Dictinary 7](#_Toc106115263)

[7. References 7](#_Toc106115264)

My Individual project will be a website for a competitive sliding puzzles. I will achieve this goal with TypeScript React for the front end and Java SpringBoot for the back end. I will be stating my reasons of choosing these languages later this paper.

# 1. Idea origin

A picture containing text, sign, several

Description automatically generatedOne of my colleges at the group project said that my game idea sounded like an already existing game called Rush Hour. Before he talked about it, I never heard of it. So, I picked it up at a local toy store for 23 euro. Even though they are both sliding puzzles they are not the same kind of sliding puzzle as Rush Hour is more based on 1 direction either up and down or left and right. That system has its uses. As this kind of sliding puzzle is easier to produce and has less chance to break. As every piece has just one way they can go when the pieces are put on the right spot. But my idea is more based on the sliding puzzles from Professor Layton. You can see two puzzle here.   
Puzzle 58: “Get the ball out! 1” from “Professor Layton and the Curious Village” (Picture left) and Weekly Puzzle 18: “Blocked In” from “Professor Layton and the Unwound Future” (Picture right)

These puzzles are more 2 dimensional as they can freely move anywhere on the board if they do not hit a wall or another piece. And as these puzzles were my real inspiration, I will be making this kind of puzzles. Because I these were my favourite kinds of puzzles in that series… As it was something even a small child could solve. Even if the way of solving was more trial and error…

# 2. The Game

For the Game itself I will be using a kind of grid system. At least when creating and at first rendering the gameboard. I will be using something I call line-blocks as it will tell me how many blocks the line must go until it hits the end.   
I did this because if a board is for example 10 by 10, I do not want it to be the same scale as 20 by 20 as I want to use the same size board. As I want to eliminate the chance that the game will take more than a screen of space. As that will disorientate the player and that is not good. That is why I am going to work with these line-blocks. As I can use those line-blocks to calculate the scale for the sprites and because I am using Pixi.js I can easily move the sprites individually. That is why I said “kind of a grid system” as it will not use the same restrictions as a grid system. As the player can move the pieces of the grid anytime they want. If it does not hit another piece or the walls.   
Hitting another piece or the wall does not cost a turn to be spend. But letting go of a piece from the piece position will cost a turn. And time will start from first interaction of a piece. Not when you just hover over the piece, as some people like to fake move to think about a possible solution of a puzzle.

# 3. Service-Oriented Architecture

RESTful API is based on 5/6 architectural constrains. Those being: “Uniform interface”, “Client-server”, “Stateless”, “Cacheable”, “Layered system” and the optional one being “Code on demand”.  
What those mean I will talk about now.

The Uniform interface is based on the idea that one resource should only have one URL linked to it. So, a URL for “/headgear” and the one for a “/headpiece” should not be separate. But combined. At least if you use it for the same reason. Like in this example where they both were looking for hats. It also means that you should use the Get, Push, Put and Delete methods correctly to minimise confusion.

The Client server is the most straightforward. It is just “Please separate the frontend from your backend” And when it says that it means that the only way for the two to communicate is by the URL’s. You can see that as someone sending emails to someone from across the country.

Stateless is all about forgetting the previous URL requests. As in the past the requests were remembered by the system. But with REST they will be forgotten and that is helpful for the fact that you do not have some random data ruining your request because you forgot to remove something.

Cacheable is the act of caching important data and responses that the system gets and sends. As that information can become useful for when you scale your project. This is done to improve performance and of course scalability.

What is meant with Layered system is the idea that you could split the backend to even smaller backends that all did one thing and one thing alone. So, one could handle the URL requests, one could interact with the database while the third one could see if the request is even legal in the first case. It is a very good thing and helpful when you do not want to be fully compromised when one server goes down as those 3 systems can be run by different servers. Microservices are a modern example of this idea. As it splits the backend in smaller parts so that you can reuse code for different situation. So, for example. You can reuse the user container of a game website for a website about puppies.

The last of the constraints is optional like I said before and its about code on demand and its about sending pieces of code back to the frontend if they request it. Maybe you had a UI thing stored in the backend or something. But of course, you must be careful with this as it can be exploited. (restfulapi, 2021)

The reason why I talked so much about the REST API is because I was debating to myself between the REST API and GRAPHQL. GRAPHQL is another kind of architecture made to “simplify” collecting data. But with the scale I want to do things it is like cutting a slice of bread with a tuna knife. Yes, you can do it. But it is way too much. Also, because my pretty small scale the system does not have too much URL’s that it has to check. Because even if I split the records, users, and the boards/worlds into small separate services it will just be easier to keep track of URL’s and the right paths then to do a client-side query to get all the information you need.   
  
Of course, there is also SOAP, Rule-based, data-centric, Plug-ins, Microservices Architecture and many more. But for now, I will be sticking with REST. Because I just like the idea of REST the most.

3.1 Model

Timeline

Description automatically generated with medium confidence

# 4. Programming languages

## 4.1 Angular

After some thinking and going back and forth on Angular and React. I have decided to go for Angular as the system it uses for html templates looks a bit better and cleaner than what you can do with react. With Angular I can just put the condition of the children appearing in the parent. And it will check it. In React you must put it in curly brackets. I also like the way it uses references. As you just have to say # before the name you give it. So, you do not have to declare the reference. It will just declare it. It is also good that Angular is TypeScript native. And not JavaScript first and TypeScript second. And the reason why I really want to use Typescript is that I want less problems showing up during production. I want them happening during development. What I mean by that is that I do not want those vague errors that the browser and JavaScript gives you. I want to get yelled at by Typescript because I forgot to make it clear what kind of data this constant could be. It is also just better for me because of how it uses interfaces to make sure only the right kind of information goes through components. So, a check if a string is really a string and not a number is good. That is what Typescript is for me: Making sure that I do not do stupid things in development.

4.1.1 PIXI.js   
But Angular will not be the only thing I will be using for the frontend as the game will be using the 2d game engine PIXI.js and its extension BUMP.js for hit detection.  
in PIXI.js there is a certain function called “drawPolygon” and that will be the main method I will be using to draw all these pieces. I will also be using the drawPolygon function to draw the walls around the current game. The pieces will be interactable, but the wall will not have the mouse events set up to do so... There are other 2d engines for Typescript and JavaScript like PhotonStorm, MelonJS, Kiwi.js and many others. But the reason why I am choosing PIXI.js is simply that when I found it, I just knew this was the one I was going to use. It was not some big decision; it was just a certain feeling I had when I saw it. With Pixi.js you can also do simple collision detection by looking for the height and width of the current rendered sprite. Sadly, there is no easy way to do hit detection for things like a L-shape. But that is more because you must calculate the shape of that object then and that is still a bit too much for my skill level. And more because it still needs to be moved easily. And things like changing the vortex data of the sprite/graphic will break the interactivity of a sprite/graphic.

## 4.2 Java SpringBoot

As how I am going to use my backend. Its will be used as a kind of highway between the frontend and the database. As most of the gameplay stuff must be done by PIXI itself. But what I mean by that is that the backend is handling the CRUD requests, but also things like the search. As people can look for certain words in the name of the puzzle. Because sometimes the names of the puzzles can be weird. As the admins can allow a random word generator to make suggestion. So, you can have a level called: “Hasta La Vista Case” or “The Rumbling Case”. The Random word generator API will just give you a list of words you can use. So, you can make the title make a bit of sense. It is also done like that because of maybe offensive titles. As I do not want them in my puzzle website.   
Of course, you can also look for the “World” names. Just like things like the Splatoon 2 Octo Expansion has the separate lines of levels. In that game the levels have the normal names and a World/Metro line number. So, for example “Matchmaker Station” is Line F stop 7 and “Outrageous Station” is Line J stop 7. Of course, I will use other words. So, Line can be Briefcase or building. But Splatoon 2 had a in game reason why they used the words used for the Metro, as you had to take one to get to the levels. So, for this explanation I just used the terminology they used/ would have used.   
Another thing I see the backend doing is a chronological list of how the world record was lowered. As it involves the task to ignore times that are lower than old times, but that were never world records. As the database will just give you the lowest time. No matter the World record status.  
So, for example: a time of 1:26 was set yesterday and now 3 new records are set in this order. 1:15, 1:21 and 1:14. What I want it to do then is get me the 1:26 from yesterday and then 1:15 and 1:14 of today. As they both were World records for a time. The 1:21 was never a World record. So, it will not show up in that list.

And when it comes to usernames and passwords that will be handled by a security program like Auth0 and I just take the ID they give to the frontend when first registering so that I can log in through their systems. As you need a way to link the two sides for the login to work.

## 4.3 SQL

For Databases will be using PostgreSQL. As I thought I would need to use a lot of arrays in my program. And as PostgreSQL has the possibility to store arrays without a lot of complains. And the reason why I did not use something like NoSQL is because the last time did not go so well when I used it. As I knew 0 things about NoSQL, and it showed. So, I am still a bit scared to use it again. It is also that NoSQL is a bit to messy for my taste. As it will just store anything without really a clean way to see where in the horizonal line it stored that one object.

# 5.Requirements

## 5.1 Functional Requirements + MoSCoW

|  |  |
| --- | --- |
| Admin | |
| FR-1 | The admin should be able to give people admin roles id they got the power to do so. |
| FR-2 | The admin should be able to remove records. (If they got the right permission) |
| FR-3 | The admin should be able to make and edit new puzzles in their puzzle editor |
| The admin should be able to name and rename the puzzles. |
| FR-4 | The admin should have the option to get random words from a generator. |
| FR-5 | The admin be able to make and edit “Worlds” |
| FR-6 | The admin be able to put the levels in “Worlds” |
| FR-7 | The admin be able to change the “World” the level is in. |
| FR-8 | The admin be able to remove levels from “Worlds” |
| FR-9 | There should be an editor to make new pieces |
|  | You should have the option to choose if the piece has any directional variants. (4 variants, 2 variants and no variants) |
|  | You should have the option to colour the piece in whatever colour you want. |
| FR-10 | There should be an editor to edit the pieces. |
| FR-11 | There should be a warning when you do this if the piece is already used somewhere. |
| FR-12 | The edited version of the piece will also affect the variants. |
| User | |
| FR-13 | The players should see their records in their user page. |
| FR-14 | Players should be able to change their nickname and password |
| General | |
| FR-15 | The User and Admins should be able to log in and sign in |
| FR-16 | The players should see the top 50 records on a certain puzzle. |
|  | If the player is below 50 then there will be 51 records. It will say their place on the leaderboard |
| FR-17 | The players should be able to see the progression of the world record in a kind of chart |
| Game | |
| FR-18 | The Game should be playable on Desktop |
| FR-19 | The Game should be mouse controlled. (Drag and drop) |
| FR-20 | The Game should track your turn count and your time. |
| FR-21 | The Game timer should stop at the moment the right piece is on the end point |
| FR-22 | The Game should only record your new time and turn count if you have logged in. |
| FR-23 | If the mouse isn’t in a certain radius of the dragged piece, it will be counted as dropped. |
| FR-24 | The game has a redo turns button. |
| FR-25 | The Game should be able to be played on mobile |
| FR-26 | The game should have a turn limit mode |

|  |
| --- |
| Must Have |
| Should Have |
| Could Have |
| Would Have |

## 5.2 Non-Functional Requirements

|  |  |
| --- | --- |
|  | |
| NFR-01 | The product should use the CI/CD |
| NFR-02 | The Product should have version control |
| NFR-03 | The loading of a page should not take more than 0.75 second. (When plugged in an ethernet cable) |
| NFR-04 | The Product should use a Distributed system. |
| NFR-05 | The product should be easy to use for the User. |
| NFR-06 | The Database should not take more than 0.5 seconds to give the required data. |
| NFR-07 | The website should use Auth0 to secure the users passwords |
| NFR-08 | The Website should have error pages it gets sends to if the system finds a big error. (So, if something completely breaks or loading takes more than 1.5 seconds too load) |

# 6.Dictinary

|  |  |
| --- | --- |
| Trickling | Another word for Sliding |
| Enigma | someone or something that is mysterious or puzzling |
| DS and 3DS | Handheld game consoles made by the Japanese company Nintendo |
| line-blocks | How many blocks to one side a line goes. Done like this because of the unknown scale of the boards. |

# 7. References

restfulapi. (2021, September 27). *REST Architectural Constraints*. Retrieved from restfulapi: https://restfulapi.net/rest-architectural-constraints/