EÖTVÖS LORÁND UNIVERSITY

DEPARTMENT OF PROGRAMMING LANGUAGES AND COMPILERS

People control the music voting based live queue music player

Consultant:

Dániel Balázs Rátai

Ph.D. student

Author:

Kristóf Kovács

BSc student

Budapest, 2022

Table of contents

1 Abstract

Songs played at parties matter because they are the backbone of the event. If we take a look at who controls the music at parties, we find that no matter how technology changed, we, the audience, had no control over it. Why do we, the listeners, accept that?

The goal of this thesis is to implement a distributed, high-performance web service that can take the role of the DJ. The system collects the votes from the listeners while the party is going and lets them decide which songs will be played. The members of the party can scan a QR code containing a link to the site where together they can control the music. The music player has a queue, which is sorted by likes. Everybody can propose their favorite melody to be played, or rank other songs based on their liking.

In the following pages, I will describe the process of creating such software, detail my optimizations, and show the system's inner workings.

2 User documentation

The following screenshots were taken with Firefox version 101.0b9 (64-bit). In general the latest stable version of most web browsers should work.

2.1 Usage instructions for the host

2.1.1 Preparation

Before the event the host needs to set up the device which will play the music (eg. a laptop, a smartphone, even a Raspberry Pi). The device needs network access to the servers. Depending on the sound system, a jack cable or Bluetooth interface is needed for connecting the equipment.

To connect to the service please open a browser. Assuming the software is deployed on “`http://deployme.nt`”, open the link. The browser should show the following.

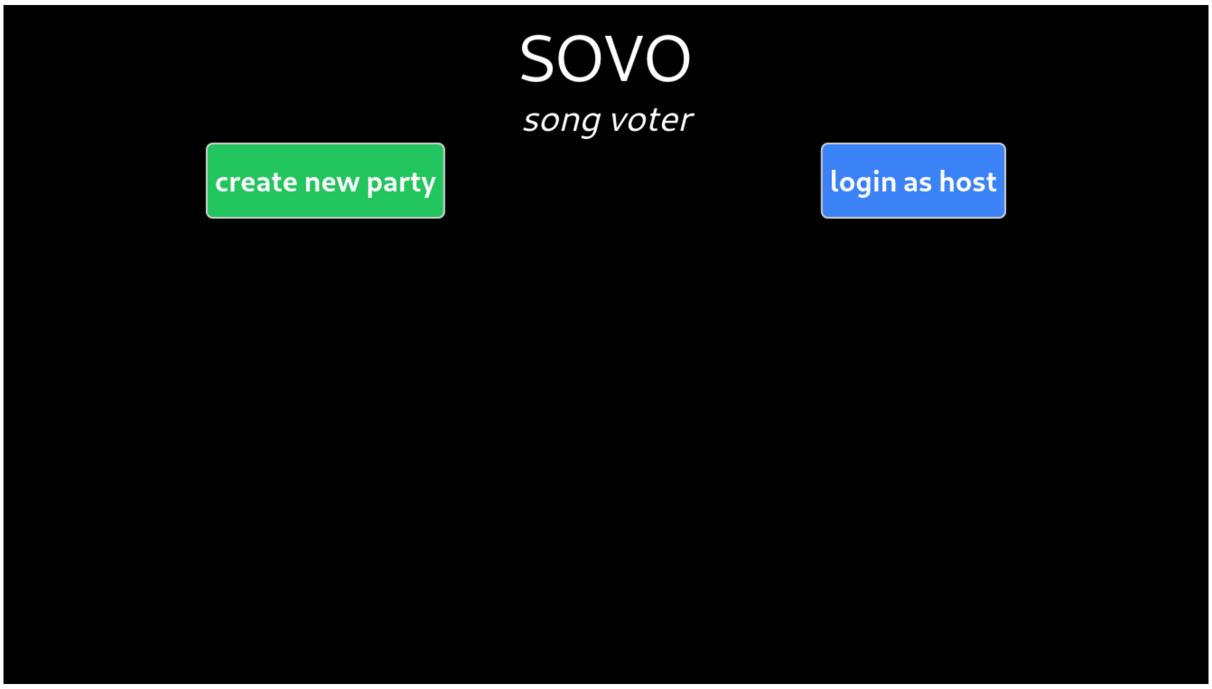


Figure 2.1 Frontpage of the system

Click on the “*create new party*” button to make the connected sound system available for the guests.

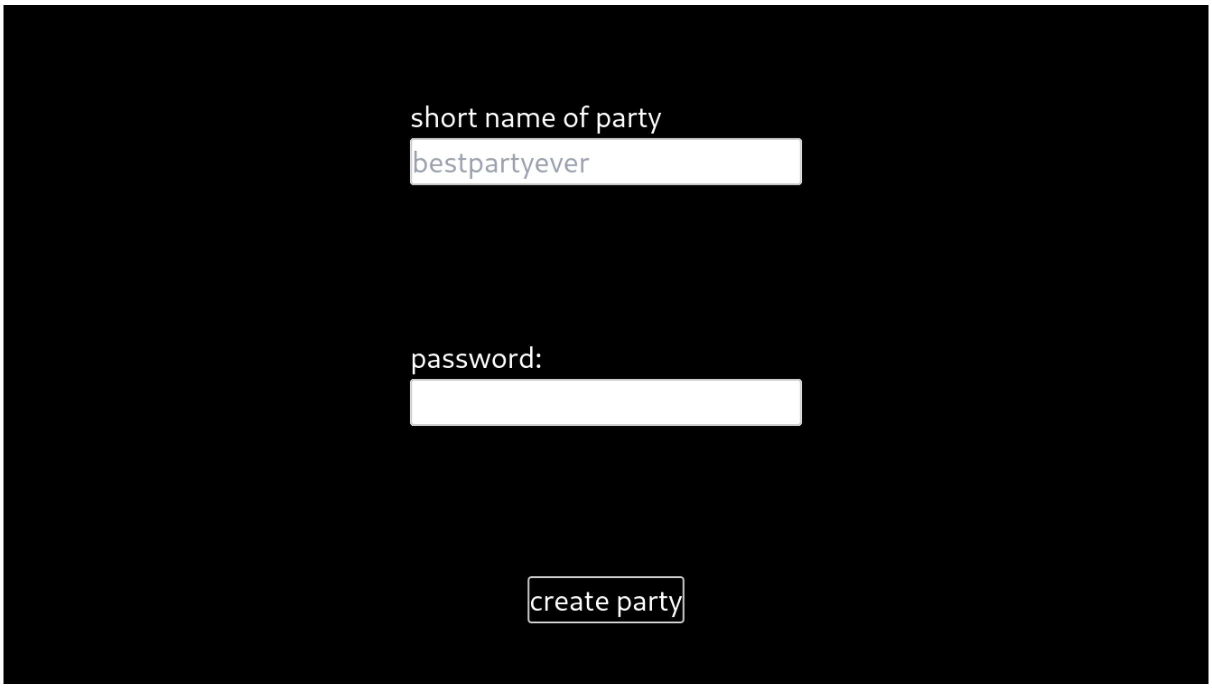


Figure 2.2 Interface for creating a new party

Please provide a name and a password for the event. The latter is needed to ensure only the host is able to control the management panel, regardless of the device. The system can handle the swap of the player device to increase redundancy, thus making it more rigid.

After entering the required fields please click on the “*create party*” button. The site should look the following.

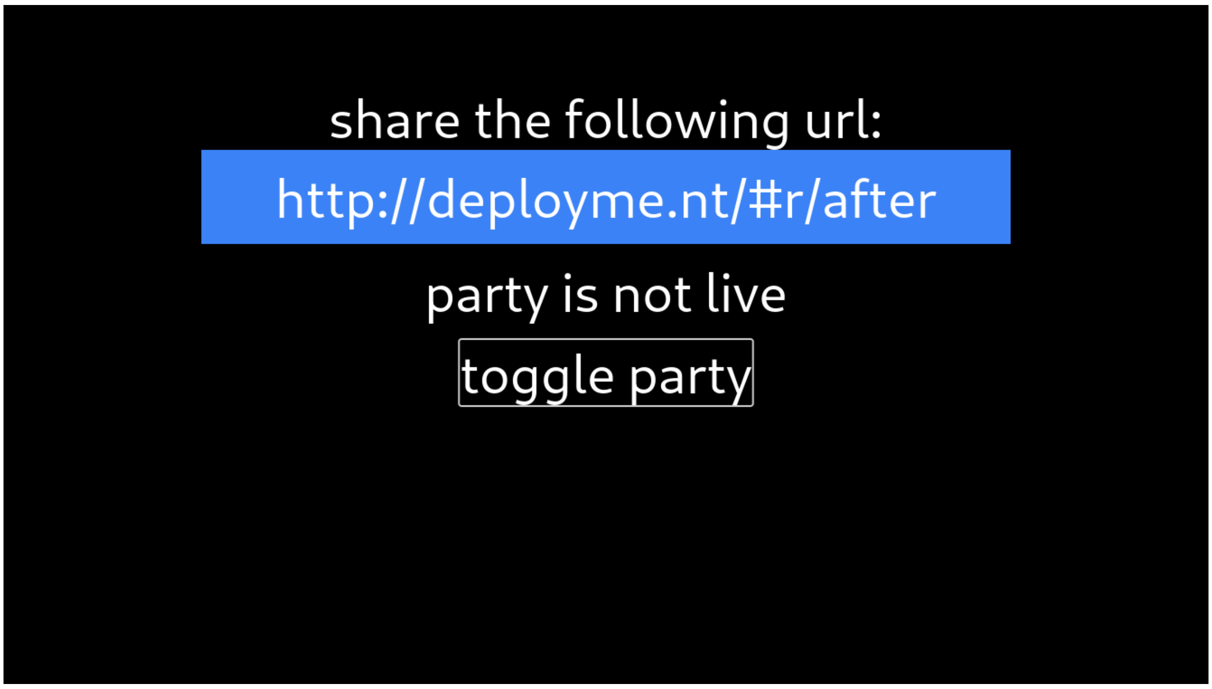


Figure 2.3 Host interface of an upcoming party

Please provide the URL in the blue box to the guests, so they can connect to the service, and propose songs before and meanwhile the event. Closing this site is safe, it will not prevent users from voting and promoting.

2.1.2 Live setup

Right before the event, the host should reopen the site, and log back into the interface. To do so, click the “*login as host*” button seen on Figure 2.1. The site should look like the following.

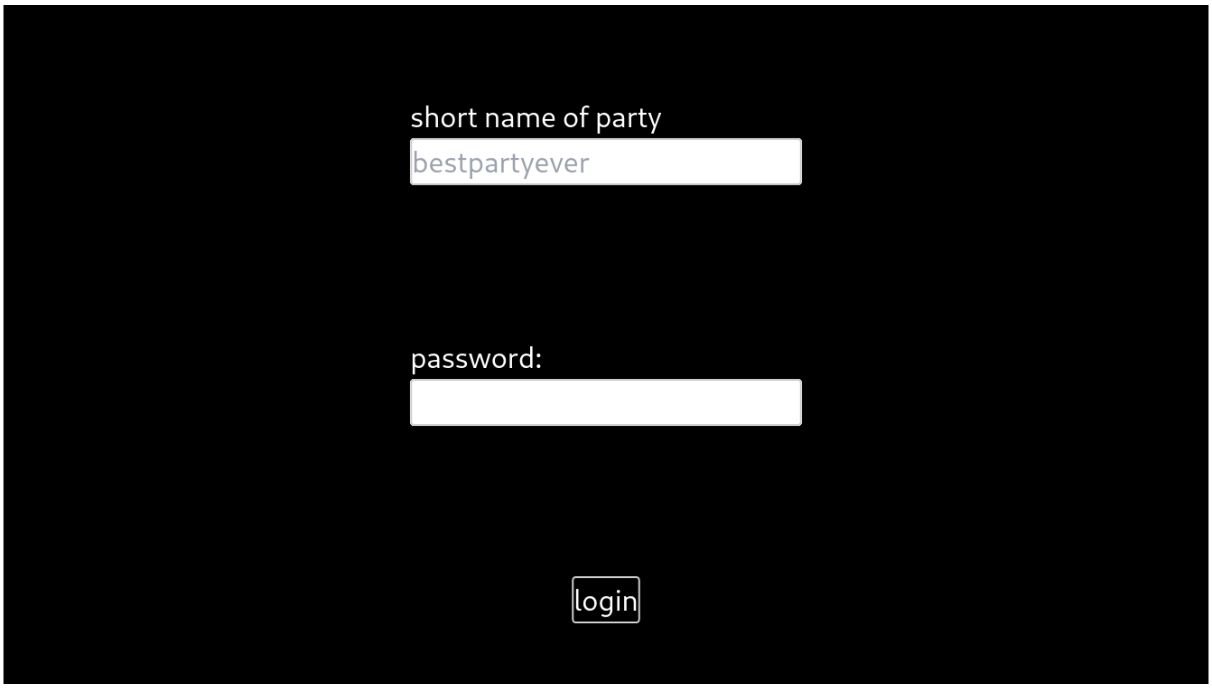


Figure 2.4 Login interface

Upon providing the correct credentials, Figure 2.3 should appear. To start and stop the player, please click the “*toggle party*” button seen on the site. After setting the party live, If the queue is not empty and users have promoted some music, then the device should start playing the most voted song.

Note: The browser might ask for permission to autoplay audio on the site. Please choose an option where audio is allowed.

The browser should show the following.

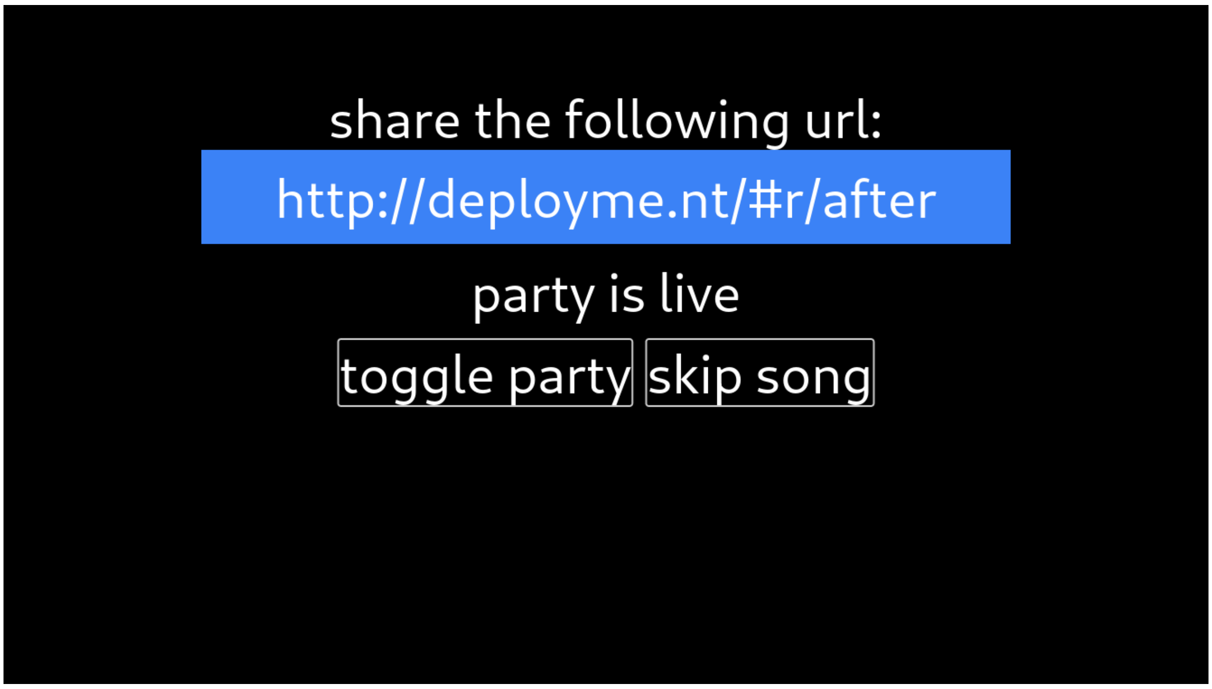


Figure 2.5 Live party interface

By clicking the “*skip song*” button, the host can command the system to stop the current melody and instead play the next most popular one from the queue.

From setting the party live, the device does not need supervision. The software may recover from temporary network problems.

2.2 Usage instructions for guests

To use the service, the user needs a network connection as well as a device capable of opening websites. The site is designed for smartphones in mind, so this is the recommended choice.

Opening the link provided by the host may show the following.

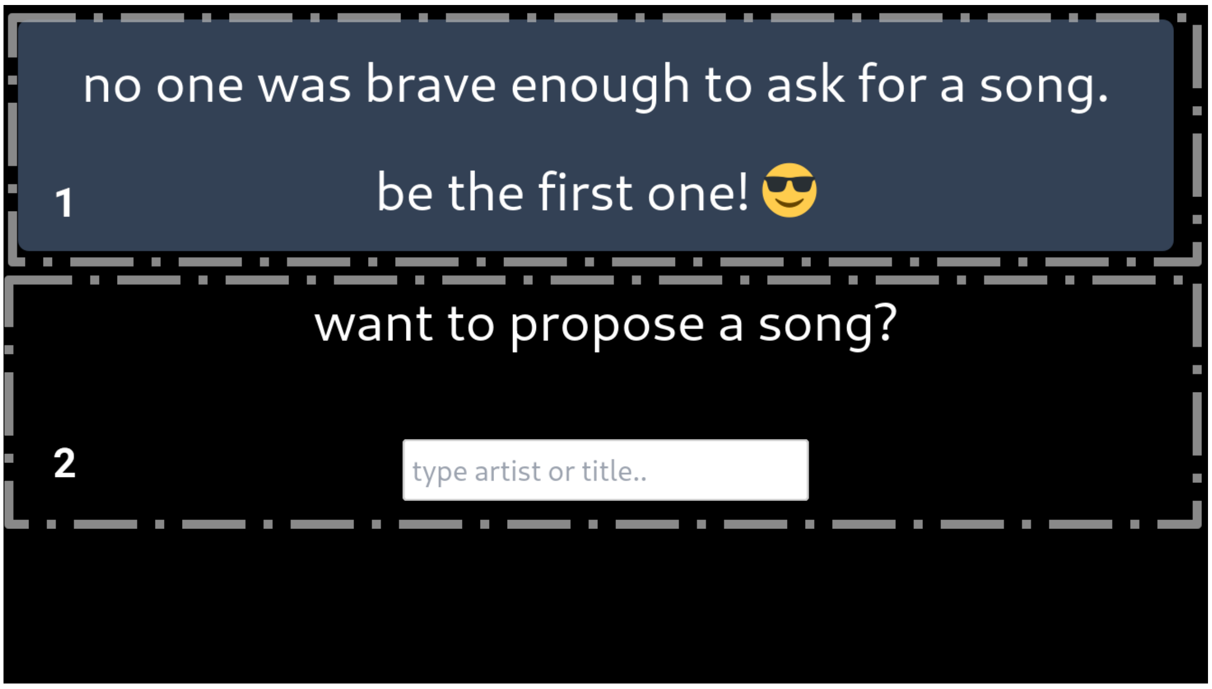


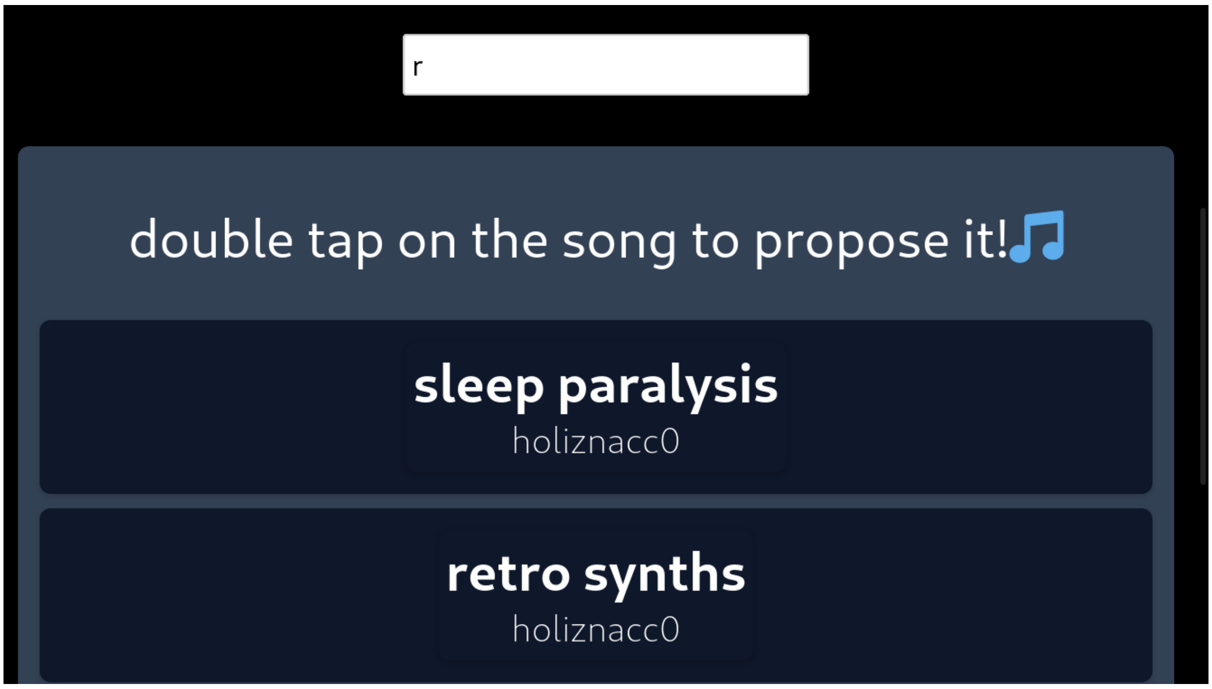
Figure 2.6 Guest interface of a party before the first interaction

On the screenshot, you can see the two main parts of the interface:

1. the queue, which is empty because no one has suggested any song yet.
2. the search box for proposing music to be played.

2.2.1 Proposing

To add a selected song from the music catalog, type in the title or the name of the artist of the chosen song into the "*type artist or title..*" field shown on Figure 2.6. While the user enters the search term, the results update after every keypress.



r

double tap on the song to propose it!🎵

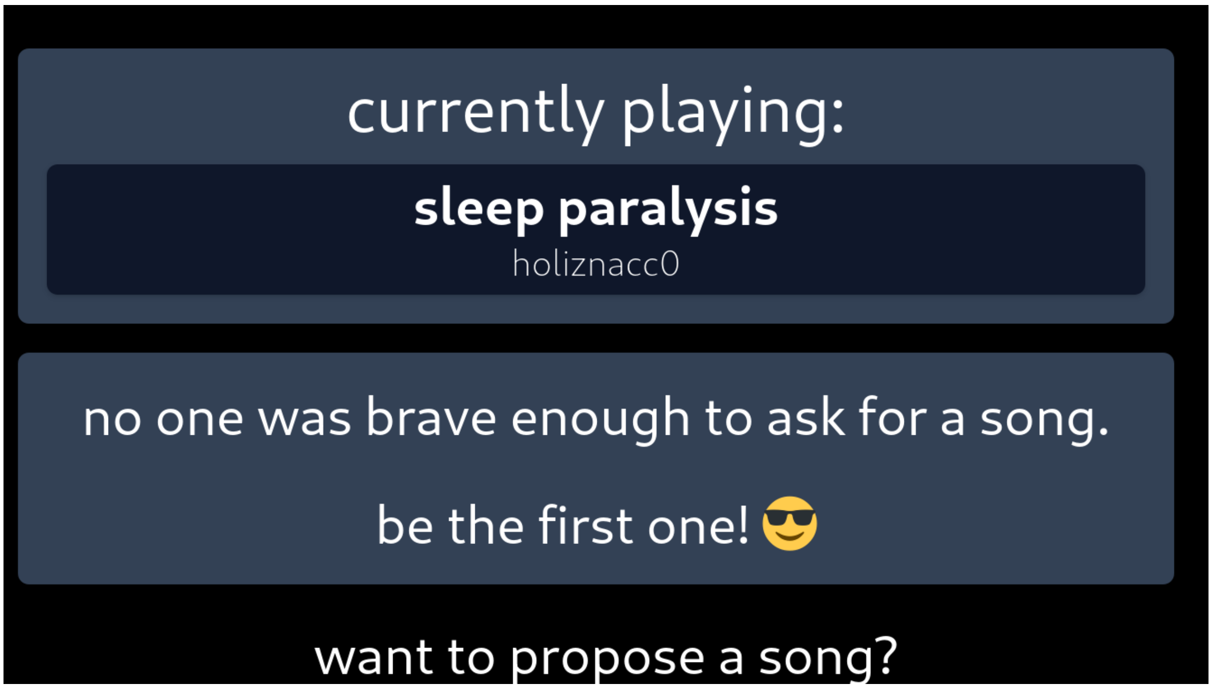
sleep paralysis
holiznacc0

retro synths
holiznacc0

Figure 2.7 Searching for songs to be proposed

2.2.2 Currently playing

Depending on the party, if it is live, and the queue is empty, then the melody gets played instantly. This happens in the following example. If the user double taps the song titled “sleep paralysis” by “holiznacc0” [1], then it gets played right after. The interface shows it in the following way.



currently playing:

sleep paralysis

holiznacc0

no one was brave enough to ask for a song.

be the first one! 😎

want to propose a song?

Figure 2.8 Interface showing the currently played song

2.2.3 Voting

Adding more elements to the queue looks the following.

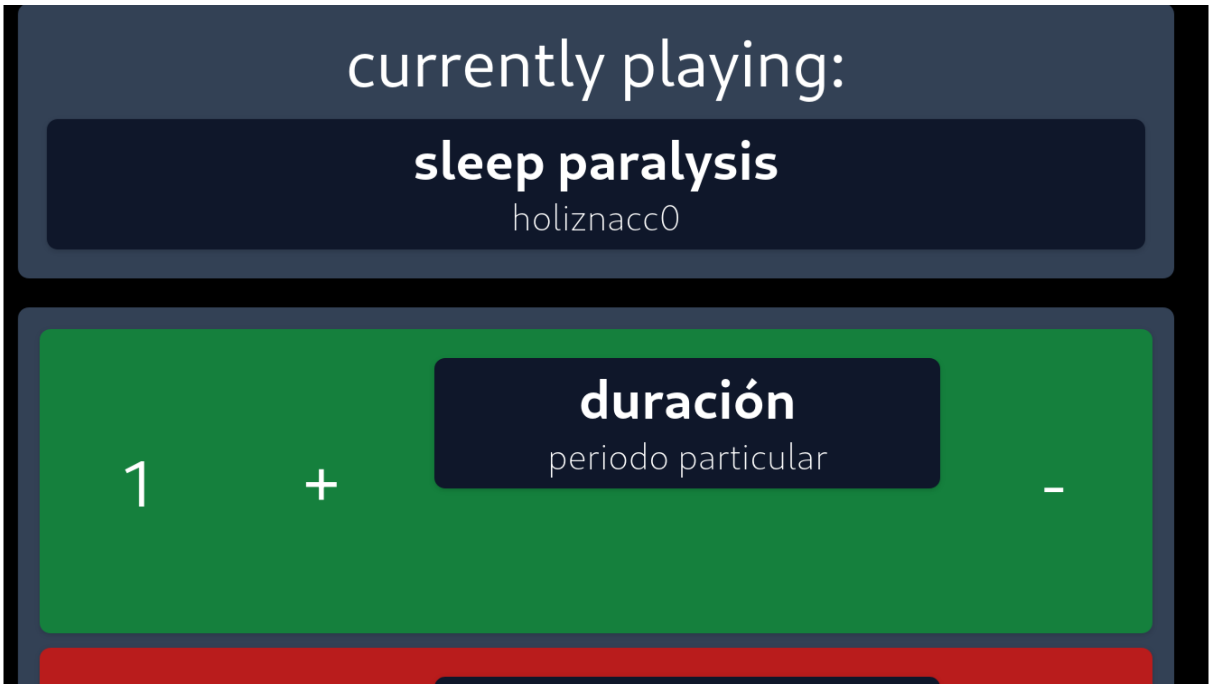


Figure 2.9 Queue with liked and disliked elements

The songs in the queue are put in order by their score, which is calculated as the number of likes minus the number of dislikes. By liking songs, the user can give a song a higher chance to be played. To do so, one has to click on the "+" sign next to the selected melody. If done successfully, the element should turn green like shown in Figure 2.9. Otherwise if the guests do not want to listen to a certain proposed song, they can declare their opinion by tapping on the "-" sign, to decrease the song's score.

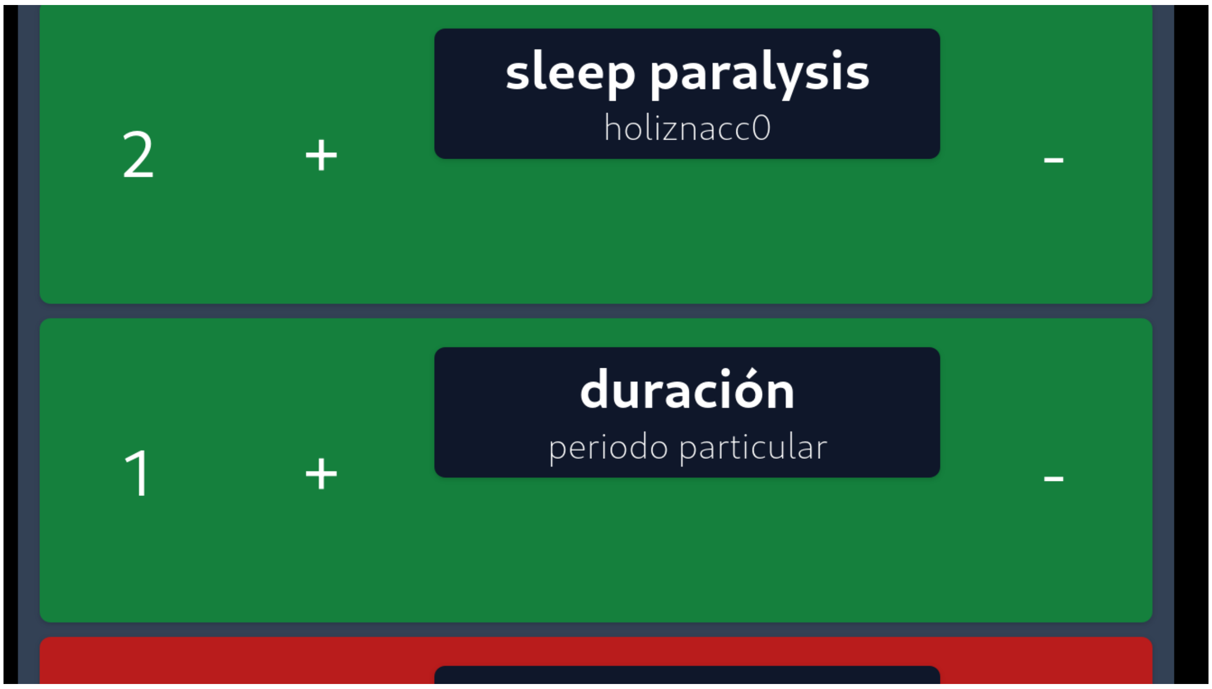


Figure 2.10 Queue ordered by rank

As more people start voting given the wide enough music selection, the played songs should approach the group’s common mainstream taste of melodies.

3 Developer documentation

3.1 The stack and environment

Telling computers their task is hard. Each of them works differently, runs various Operation Systems (if any), and has different components. For solving those inconsistencies, programmers rely on codebases which help them develop their products faster, and make them easy to use on every device. I have also used various libraries and technologies to develop this software. Usually, those frameworks can be categorized by the task they help to solve. Those and other programs create the toolset to develop and maintain such systems.

In the following chapters, I will describe which tools I have chosen, and why I switched from the previous ones. Each section will focus on particular tasks, like creating pages for guests, receiving votes, and storing the audio files. I get into detail about the reasons why I did choose my implemented technologies.

3.1.1 Frontend

This is the control panel for every user. The goal is to create a fast and easy-to-use interface for guests and the host.

Websites are constructed with HTML (HyperText Markup Language) which defines the structure of the site with nested elements represented by tags. Then with CSS (Cascading Style Sheets), the developers can enhance the styling, and with scripts running in the browser (written in JavaScript, or WebAssembly) sites can be interactive.

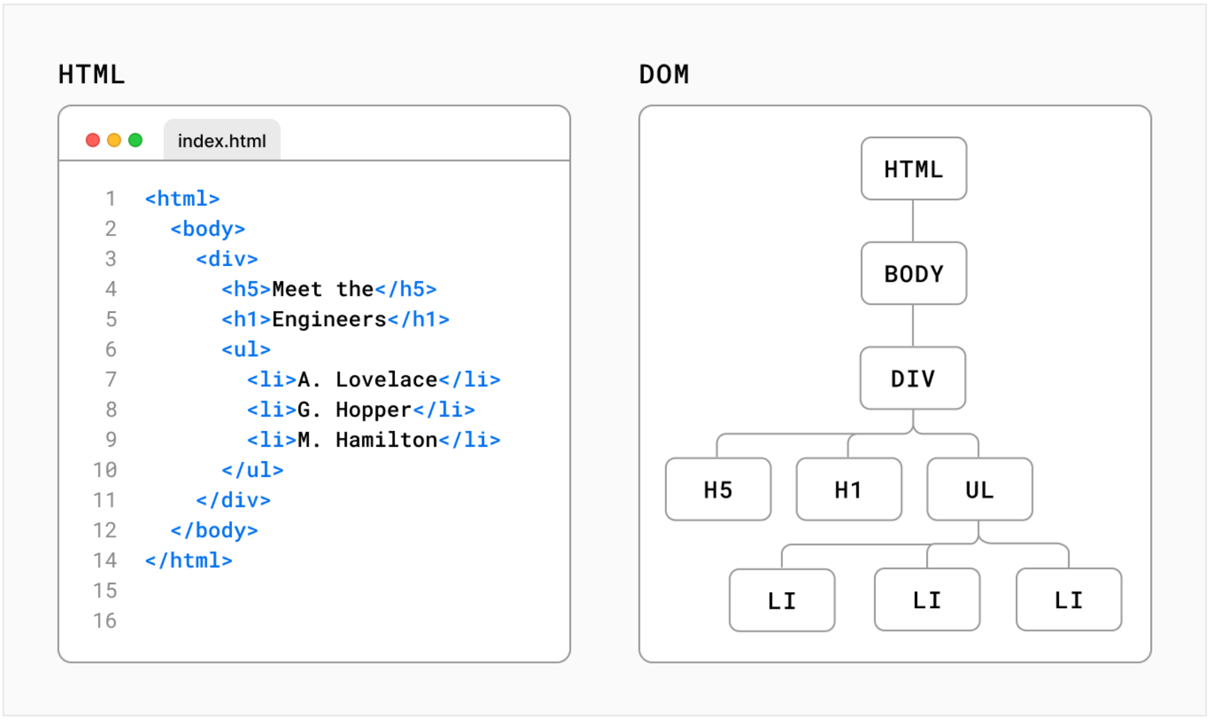
There is no perfect language or tool to solve every single problem. Every language has great properties that make them a good choice for certain use cases, but they also have flaws that make them a bad and inefficient solution for others. The goal of the programmer is to utilize the tools and construct those systems to be fast and secure. In the following, I get into detail about the differences between Javascript frameworks for constructing frontends.

3.1.1.1 Native Javascript

Every site can be created without any frameworks, with just Javascript, but it has some drawbacks. Without the help of frameworks, programmers may develop features slower and can have a higher risk of getting into antipatterns, which may result in the system being hard or expensive to maintain and expand. Not all features are supported in every browser version [2]. Frontend frameworks, by design optimize for accessibility by purposefully using features widely supported by most browsers [3].

3.1.1.2 Angular

Sites are represented by components embedded in each other. This is called the DOM (Document Object Module). The browser interprets the HTML document sent by the server, initializes the DOM then shows it to the user.



```html
<html>
  <body>
    <div>
      <h5>Meet the</h5>
      <h1>Engineers</h1>
      <ul>
        <li>A. Lovelace</li>
        <li>G. Hopper</li>
        <li>M. Hamilton</li>
      </ul>
    </div>
  </body>
</html>
```

Figure 3.1 HTML code parsed to DOM [4].

Interactive frontends manipulate the DOM by adding, deleting, or changing elements of it.

One of the earliest and most widespread frontend frameworks is Angular [5], whose predecessor, AngularJS has been developed since 2010. Originally, It started as a side project by a Google employee, and since has grown to be one of the biggest open source application frameworks. Compared to other stacks like React, Angular has a "*batteries included*" approach, meaning, it has libraries included for solving a wide variety of problems. It has a testing framework, a routing library and a style preprocessor pre-configured by default. Because of this, Angular projects are considered oversized.

3.1.1.3 React

Most frontend frameworks work by creating their own simplified copy of the DOM - called Virtual DOM [6] - and render the site based on it.

React [7] is a frontend library for creating reactive websites. The project is developed and maintained by Meta (formerly Facebook). It is the most popular frontend framework [8] currently on the market. The project has been open source since 2013. The library focuses only on state management and updating the DOM via diffing its internal Virtual DOM [6]. Thanks to its wide and active community, with the help of libraries development can be fast and precise.

3.1.1.4 Vue

Vue is a lightweight frontend framework inspired by AngularJS [9] developed since 2013. The creator of the library was not satisfied with the size and complexity of AngularJS, and tried to make a barebone framework which only focuses on the view layer.This framework is similar to React because both use a Virtual DOM [6].

The first prototype frontend of my voting system was made with it, but I switched because of lack of community support.

3.1.1.5 Svelte

Every computer is an interpreter. It gets instructions in machine code and executes them. There are two major groups of programming languages. In every case the instructions are translated to the language of the machine.

One group, the interpreters just take instructions one after another, execute them, then move forward to the next step.

The other solution, compiling is based on looking some steps ahead, and trying to optimize the work. Those programs, which try to translate to more effective instructions are called compilers.

In general, compilers are complex pieces of software, because they have to parse the source code, create byte-code or Intermediate Representation[10] of the software based on input, and then construct the corresponding executable program.

The third category is hybrid, meaning a compiler translates to a language which will be interpreted by an interpreter, a virtual machine. Every framework in the frontend chapter does it one way or another, because they translate to Javascript, which is then interpreted by the browser.

To be exact, Javascript was only interpreted until 2011, because afterwards Internet Explorer 9 was released with a new engine named Chakra, which was basically a virtual machine interpreting the byte-code compiled at runtime, so Javascript became a JIT (Just-In-Time) compiled language [11].

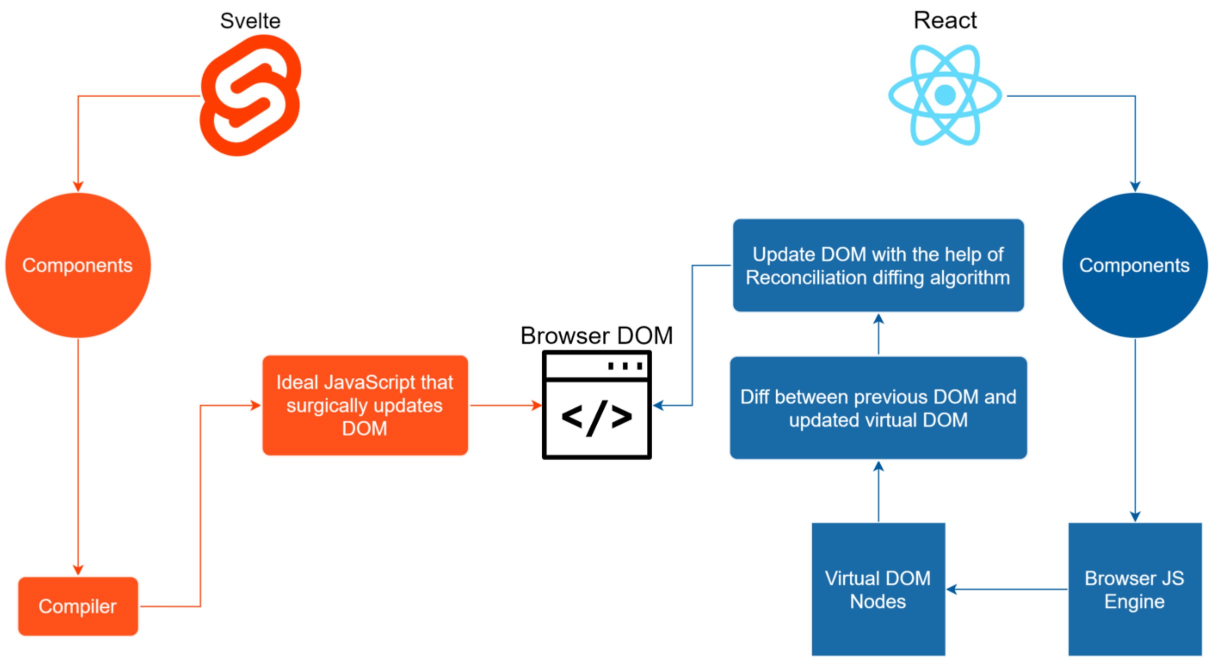

Svelte
React
Components
Update DOM with the help of Reconciliation diffing algorithm
Components
Browser DOM
Ideal JavaScript that surgically updates DOM
Diff between previous DOM and updated virtual DOM
Compiler
Virtual DOM Nodes
Browser JS Engine

Figure 3.1 Difference between Svelte and React DOM update method [12]

Every solution on the spectrum has advantages and disadvantages. Having a Virtual DOM is fundamentally one more layer of intermediate representation of the work. To gain performance, Svelte tries to be more like a compiler, and does DOM operations directly, reducing memory footprint and computation complexity, thus achieving greater performance.

Unlike Angular, Svelte is more like a frontend toolkit. To solve problems like routing between sites, testing, - which are handled by Angular out-of-the-box -, the developer needs to choose their own solutions. As every complex decision, this also has two sides. Everyone has to choose.

Svelte was one of the tools that helped me create the frontend of the system.

3.1.2 Static file server

Storing and serving data is a critical part of applications. When the content is not subject to frequent modifications, the data is usually distributed in files. In our case, one simple server will provide static files to the clients, but enterprises and companies use a CDN (Content Delivery Network) service to deliver their data all over the globe.

3.1.2.1 nginx

nginx [13] is a multipurpose, high-performance web server. It's used by Netflix, Pinterest, CloudFlare and Airbnb to name a few. [14].

One of its features is it can function as an API gateway, and also serve static files, which is ideal for our use case. One problem with nginx is that the setup needs a configuration file.

3.1.2.2 sirv

The foundation for building the frontend was Svelte’s official template. The configuration utilized sirv [15], a command line tool with a better developer experience than nginx. With command-line parameters, and one command, a whole compiled project can be set up, compared to nginx, where a configuration file is the easiest way to host the assets.

3.1.3 Database

When data changes frequently or the service needs to serve statistics in a dynamic context, databases might provide a better solution because they can store data more effectively. With indexes, recalling or manipulating data is faster at the cost of more space requirements. Databases can store their information in many ways such as:

- key-value pairs (Reids [16])
- column (PostgreSQL [17])
- document (MongoDB [18])
- graph

Choosing the right technology is important, because it can greatly impact the cost of operation, and maximal load of the system.

3.1.3.1 PostgreSQL

Most of the currently operating databases on the internet are relational. Data is collected into tables, where every column has a type. Each record has to follow the schema of the table. Tables can be logically connected, hence the name relational.

Many solutions use SQL or some dialect of it: MySQL, SQLite, MariaDB, and Oracle SQL to name a few. Some of them are closed source, meaning that developers cannot check the source code of the engine. PostgreSQL[19] is the most advanced open-source SQL database with over 30 years of active development. For the first prototype, I was considering PostgreSQL for storing user data. During development I switched because relational databases can perform worse for updates compared to document centric data storage solutions.

Relational databases have some drawbacks. As the amount of stored data increases, so does the maintenance cost. Designing layouts is complex and difficult compared to document based databases.

3.1.3.2 MongoDB

MongoDB [18] is a document-based database. Data is stored in documents which are grouped into collections. Unlike SQL, the documents do not have to follow a schema, they can be heterogeneous by type. This flexibility greatly improves developer experience, but sometimes can create weird situations. Because of this, most MongoDB drivers support schema checking for various operations to guarantee consistency.

Of course there is no technology without drawbacks. Documents tend to use more storage compared to table-based systems, for storing field names with redundancy, and some operations can be magnitudes slower.

3.1.3 Backend

Establishing and maintaining communication is hard. Sometimes the two peers cannot reach each other, they have to exchange data but need to hide sensitive content. Those systems provide infrastructure that is invisible for the end user, but without it, services would not be able to function. Managing, processing and accessing data is one of the main goals of backends.

Redundancy sometimes fights against, sometimes next to the developer. For efficient databases, the redundancy of the stored data should be as low as possible. Most compression algorithms try to find repeating, redundant parts in the information, they fight against redundancy.

On the other hand, if multiple sources are available for the same data, then although it takes more space than the minimum required, the failure of one instance does not threaten the accessibility of the underlying resource.

For example IPFS (InterPlanetary File System) [20] is a data storage solution without centralized infrastructure. Every node in the system holds parts of the network.

The voting system is also designed with the same principle. With starting multiple backend service workers, not only the throughput can be scaled up, but the stability of the system can be hardened.

With a federated or centralized load balancer, the backend instances can manage requests if one node or process fails.

3.1.3.1 Flask

Showing recent information to users is not an easy problem, but has various solutions. There is always a template which is filled out with the relevant information on the frontend or backend, or basically in any type of User Interface. As tags and classes in HTML enable to mark elements, which can be modified with Javascript, so does Flask [21] use a markup language named Jinja [22].

This framework is implemented in Python [23]. One of the greatest advantages of this language is readability. Take a look at the following code.

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return f"2*2={2*2}"
```

Flask works with SSR (Server Side Rendering), meaning that the site template is filled by the server, and then sent to the client. Older stacks like PHP, and newer ones like Next.js[24] use such a technique. Rendering server-side has several benefits, such as the possibility to make the site more likely to be shown on the front page of a search - called Search Engine Optimization -, as well as loading faster on low traffic. At higher throttle, utilizing the client for rendering the site can reduce the load on the servers.

3.1.3.2 Node

The most popular solution for creating a backend is Node [25], which provides a Javascript environment, but outside of the browser, running at the server side.

Ryan Dahl, the author of Node once in 2018 said he regrets some of the design decisions which have turned out to limit the growth and efficiency of the project. The software has been developed since 2009, but in the 9 years of library and Javascript language development, at the time of the talk, he acknowledged the lack of modern features from Node, which cannot be retrofitted without complete redesign, such as:

- Promises and async functions for higher-performance IO (Input Output) operations, which is the main purpose of backends, to move data between peers.
- ECMAScript Modules which were introduced with ES6 in 2015, which has officially standardized packaging of Javascript codebases.

- Typescript, a Javascript dialect with an optional type annotation for increasing security and performance, and lowering ambiguity of data flow.

To solve those problems, the author has started building a new environment with lessons learned from Node. The new project’s name is Deno[26], a secure, high-performance runtime built on top of Google’s V8 engine, and with its core library written in Rust [27].

Rust is a compiled, low-level systems programming language, which means that with the help of LLVM [10], the compiler can produce high-performance, async operations. The language has a compiler feature known as borrow-checker, which ensures memory-safety at compile time.

This compile rule ensures that at any given time, every variable has an owner, and only one function should have write access to data. In other languages, if those rules are violated, they are called “*race conditions*”.

This feature guarantees single-threaded programs, but also async, and multithreaded processes from race conditions, which are one of the hardest types of flaws in software to find and solve.

Concurrent programming, - where multiple computations happen at the same time -, is considered to be one of the hardest fields in software development. In Rust, with fearless concurrency developers can create high-performance, high-throughput IO programs running natively with heavy optimizations.

Deno’s Rust core uses a concurrent runtime called Tokio [tokio].

3.1.3.3 Actix

Actix [] is a backend framework written in Rust[27]. The web framework builds on top of Tokio ensuring high-performance and capacity. Compared to my first prototype, where I used Python for backend, Rust has an advantage of being a compiled language, not interpreted. This enables more efficient optimizations thanks to the compiler using LLVM [10] and thus higher performance.

MongoDB provides an official driver for Rust [28]. With carefully designed systems, according to benchmarks, a backend created with Actix can outperform a Flask implementation by a time factor of 27.7 times [29].

That means for the voting system, with a high-efficiency backend and database, with the same hardware, the software may serve 27.7 times more people with the same resource requirements.

3.2 Structure of deployment

The infrastructure breaks into four main parts: GST, SRV, HST, and DB.

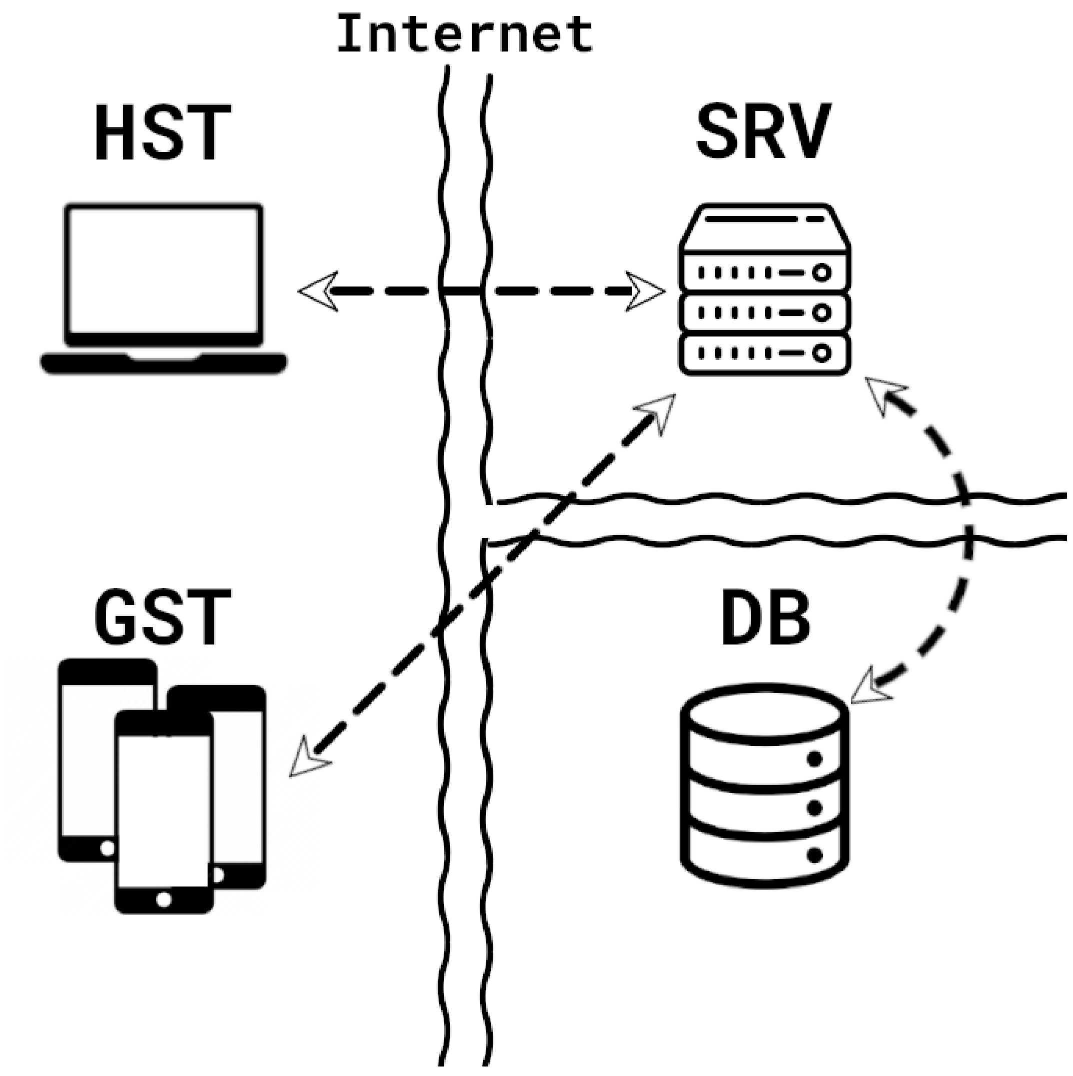


Figure 3.4 Schematics of the infrastructure

GST is the audience, the listeners. They are provided with the UI to access the service, vote and propose songs.

HST is the player connected to the sound system of the event. It has a basic UI for moderating, but it generally works by itself after setup.

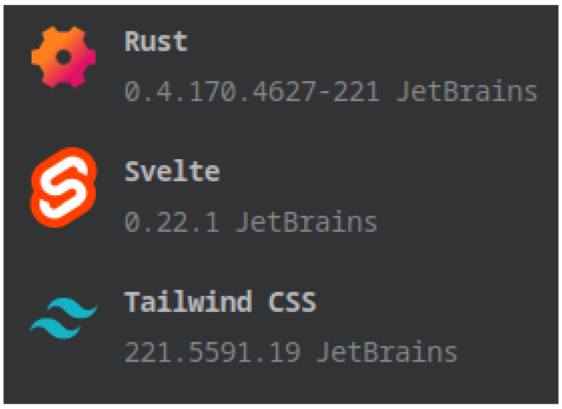
SRV is responsible for business logic, managing the queues and communicating with other peers. Note that horizontal scaling can be achieved by deploying several SRV services behind a balancer to provide higher throughput, thus more user capacity.

3.3 Setup of the development environment

3.3.1 Editor

The editor is the main tool of the developer. The most popular choice is Microsoft's open-source Visual Studio Code, which has many features built-in out of the box such as linting, Git support, easily installable extensions, and great developer experience.

My choice was Intellij IDEA[30]. I have used the following plugins.



Rust
0.4.170.4627-221 JetBrains

Svelte
0.22.1 JetBrains

Tailwind CSS
221.5591.19 JetBrains

Figure 3.3 Intellij IDEA plugins

3.3.2 Version tracking

Working together can be hard. Tracking the project's state can be challenging if modifications are not synchronized between developers.

Version tracking solutions solve those problems:

- Give a single source of truth about the state
- Manages versions
- Helps with finding the source of bugs [31]

One of the most popular tools for solving this issue is named Git [32], which is a decentralized solution designed to handle projects of various sizes. Not to be confused with GitHub which is a platform based around the former. It serves as a centralized Git node to store the data, and track community reported problems and tasks.

3.3.4 Deploying

The deployment is recommended, but not limited to Linux. The following instructions should be reproducible on any general modern Linux system (even on WSL [33])

3.3.4.1 Frontend

In general, on any system, there are only two requirements from the static file server to function with full performance:

- A stable network connection between the server, and the other three peers: GST, SRV, DB
- Serve static content with support to HTTP/1.1 Partial Content request, because the GST player does not request the whole media file for playing.

If the latter is not provided, the player may fall back to non-range HTTP requests.

Given that on a LInux system `` `npm` `` is installed and up-to-date, to compile the frontend, execute the following command in the project's `` `frontend` `` directory:

```
$ npm run start
```

According to the project's configuration the sirv [15] process should start serving the compiled frontend.

3.3.4.2 Backend

To build the executable from source, install the Rust toolchain via rustup [34]. Afterwards, navigate to the project's `backend` directory, and run the following command:

```
$ cargo build —-release
```

To start the backend, the process may need root permission to be able to bind to ports lower than 1024. From a root shell, execute the following command:

```
$ MONGODB_URI=<uri_of_db> ./target/release/backend
```

3.3.4.3 Database

MongoDB is hosted on MongoDB Atlas which provides the database from the cloud. Although using the free cloud service is more convenient, it needs internet connection, but it is not mandatory. One can host their own deployed instance of MongoDB. The minimum required MongoDB version is 5.0. With own deployment, the database should be reachable from SRV.

The backend requires a database named `` `sovo` `` on the MongoDB instance, with two collections named `` `sessions` `` and `` `parties` ``.

3.4 Implementation

3.4.1 Frontend - Svelte

3.4.1.1 App

This is the root component of the site. With a minimalistic router named tinro[35], I was able to solve routing without the need of extra configuration of the static file server, but still be able to generate URLs, which point to different parts of the SPA (Single Page Application). The library has several modes of operation, one of them URL hash [36].

3.4.1.2 Candidate

Candidate is the component which holds the elements of the queue to be voted, and sends requests to SRV, via the backend"s `vote` service. Based on previous user input, the panel can be green, which corresponds to the song being liked, dark-blue, indicating no prior vote, or red, which only happens when the user had disliked the proposed melody.

For visuals of the component, see Figures 2.9 and 2.10.

3.4.1.3 Create

Create is one of the first-level components of the component tree. It constructs the page when the user previously selects the button "*create party*". The functionality of the interface is to gather '`partyid`' and '`password`' from the user, and send it to SRV to create a non-live party.

For visuals of the component, see Figure 2.2.

3.4.1.4 Guest

Guest is also one of the first-level components of the component tree. It is rendered when the user opens a party's URL. Thanks to the nested components, the user can propose and vote songs, get updates about the queue, get to know the currently playing song.

For visuals of the component, see Figures from 2.6 to 2.10.

3.4.1.5 Host

Host is one of the most complex elements of the frontend. It is shown when the user logs into or creates a party. With the help of buttons "*toggle party*" and "*next song*", the host can control the party.

For visuals of the component, see Figures from 2.3 and 2.5.

3.4.1.6 Login

Login sits on the first-level of components from the component tree. It structures the page when the user previously selects the button "*login as host*". The functionality of the interface is similar to component Create's, to gather '`partyid`' and '`password`' from the user, and send it to SRV to log in to the party to control it.

For visuals of the component, see Figure 2.4.

3.4.1.7 Preview

Preview is a child component of Search. It only appears when the searchfield is not empty in component Guest. Shows the metadata of songs matching query criteria. Upon double tapping, the component sends a request to the `propose` service.

For visuals of the component, see Figure 2.7.

3.4.1.8 Queue

Queue is a child component of Guest. It tries to show the latest state of the proposed songs by querying for changes every 4 seconds. Renders the elements in descending order by their rank via component Candidate. If the queue is empty, then displays an encouraging motivational text to users to get them to promote songs.

For visuals of the component, see Figure 2.6 and 2.8 to 2.10.

3.4.1.9 Search

Search is a child component of Guest. It only appears when the searchfield is not empty in component Guest. Shows the songs matching the query criteria. It gives instruction to double tap on a query result element to propose it to the party"s queue.

For visuals of the component, see Figure 2.7 and 2.8.

3.4.1.10 Song

Song is the most used, versatile component of the project. Shows the metadata of the song by the provided `songid`.

For visuals of the component, see Figure 2.7 to 2.10.

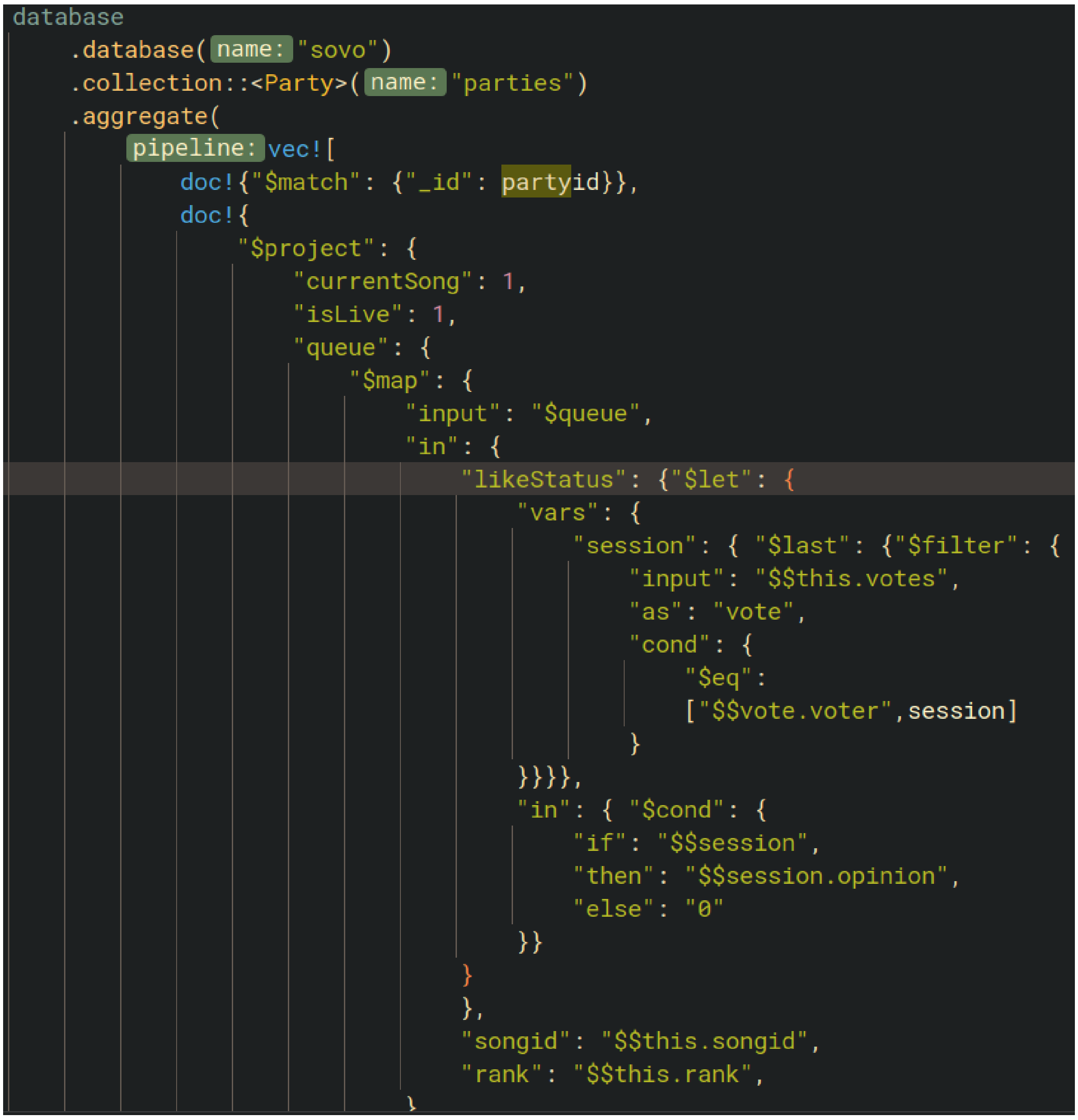
3.4.2 Actix

In this paragraph, I will get into detail about the services of the backend.

3.4.2.1 `show_queue`

The method gets a request for the queue with two parameters: partyid, and sessionid. sessionid is needed to determine if the user has liked or disliked songs. Without it we wouldn't be able to prevent double voting but those tokens are secrets of clients, that's why when requesting the queue the tokens are shadowed. After validating the inputs, an aggregation pipeline gets executed, which filters for the given partyid, then transforms the result with “`$project`”.

The password field is left out on purpose to remain secret. “`currentSong`” and “`isLive`” get sent without any modification. The queue is mapped, and every element gets transformed. For every song in the queue, the songid, and rank remains untouched, but a field called “likeStatus” gets calculated based on the vote originated from the requester’s sessionid. Without the latter, the tokens stored in the queue would be leaked, and any user requesting proposed songs would be able to vote in the name of the previous voters, by abusing their tokens.



```
database
    .database(name: "sovo")
    .collection::<Party>(name: "parties")
    .aggregate(
        pipeline: vec![
            doc!{"$match": {"_id": partyid}},
            doc!{
                "$project": {
                    "currentSong": 1,
                    "isLive": 1,
                    "queue": {
                        "$map": {
                            "input": "$queue",
                            "in": {
                                "likeStatus": {"$let": {
                                    "vars": {
                                        "session": { "$last": {"$filter": {
                                            "input": "$$this.votes",
                                            "as": "vote",
                                            "cond": {
                                                "$eq":
                                                ["$$vote.voter",session]
                                            }
                                    }}}},
                                    "in": { "$cond": {
                                        "if": "$$session",
                                        "then": "$$session.opinion",
                                        "else": "0"
                                    }}
                                }
                                },
                                "songid": "$$this.songid",
                                "rank": "$$this.rank",
                            }
```

Figure 3.4 MongoDB queue request aggregation

3.4.2.2 `get_session`

Simply inserts the timedate of the request. After the query, an ObjecId gets returned, which is sent back to the client,

3.4.2.3 `vote`

The request needs four parameters: sessionid to like the proposed song, partyid, and song_oid to know which song to vote on, and a boolean is_like, describing the opinion. Two queries are executed. The first one tries to remove previous votes if any. The second query pushes the token and the opinion to the song's votes queue

3.4.2.4 `propose`

The request needs three inputs: sessionid to like by default the proposed song, partyid to find the queue to append to, and songid to add the song to the queue

3.4.2.5 `create_party`

The request needs two inputs: partyid of the freshly created party, and password which is a salted SHA256 hash. The query inserts a new document with the given partyid, sets the party's "`isLive`" field false, "`currentSong`" to `null,` password to the salted hash value, and queue to an empty array.

3.4.2.6 `next`

The request needs two inputs: partyid and password, which authorizes that the host sends the requests. There are three queries executed. First, set the party's "`currentSong`" to the first item of queue, then removes that first element, and returns the party's stored "`currentSong`"

3.4.2.7 `toggle`

The request needs two inputs: partyid and password, which authorizes that the host sends the requests. The query finds the party with the given password, and if it matches, then sets the isLive to its opposite boolean value.

3.4.3 MongoDB

On the following figure, the database's connections can be seen.

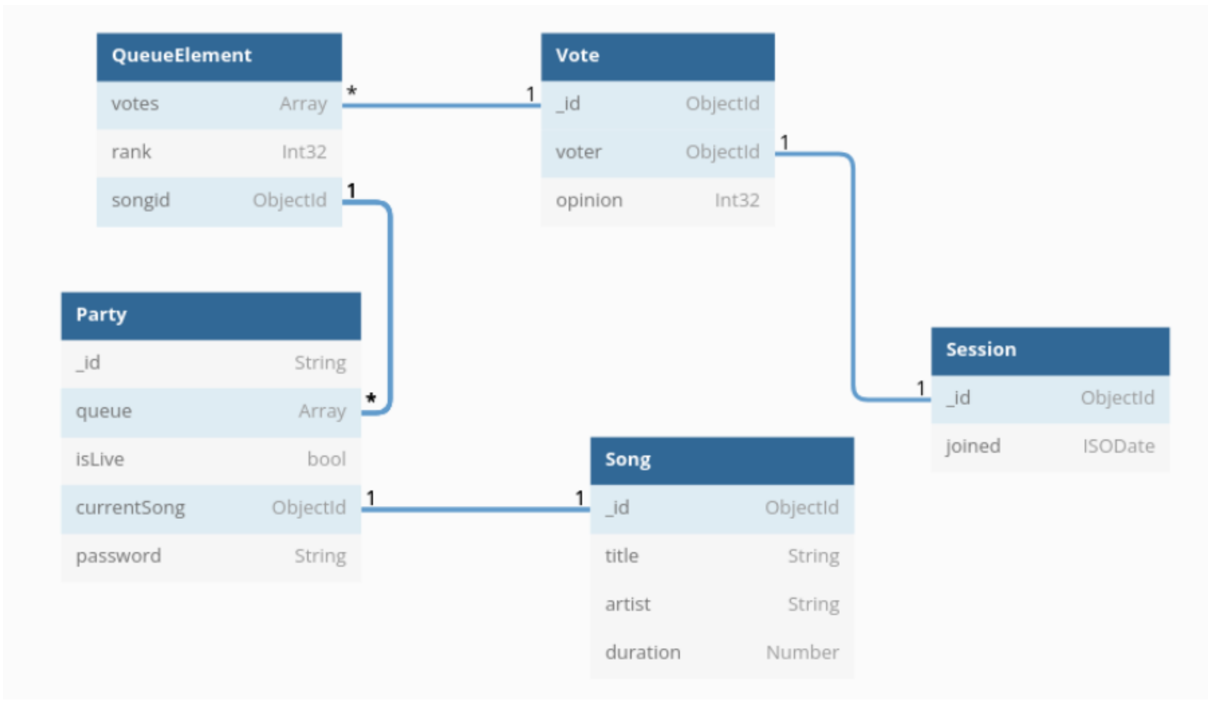


Figure 3.5 MongoDB connections between collections

3.4.3.1 `session`

Fields:

- _id - the identifier of the session
- joined - DateTime of creating session

3.4.3.2 `parties`

Fields:

- _id - the identifier of the party, shared in the URL
- isLive - boolean value controlling HST to play or not to play song
- currentSong - ObjectId containing the id of the song to be played currently if the party is live
- password - salted hash token to authorize host
- queue - Array containing QueueElements

3.5 Tests of the system

Those are black box test cases to ensure the system has the desired functionality

Test case	**Expected result**
Open the URL pointing to the static file host.	Device shows Figure 2.1
Clicking on the create new party button, entering data, then clicking on create button	Database updates with the data filled into fields, with password salted hashed
Proposing a song from a client	Every client updates within five seconds
Voting down the previously promoted song	For the downvoting user, the card turns red, for others the score decrements by two
With a non-live party and a non-empty queue, the host presses the toggle party button	After a few seconds the music starts playing
When the music on the queue reaches its end, and the queue is not empty	The most voted song of the queue gets played
Trying to log in with bad password, and try to toggle the party	Nothing happens, the party does not get toggled
Trying to create two parties with the same name	The user interface displays an error message
Entering invalid name for partyid while creating, because it contains a space	The site throws an error message to the user

With an HTTP request at vote, instead of giving a boolean to is_like, a string is passed	The backend refuses the request

Conclusion

The created software is capable of achieving its goal, to be easily accessible for anyone, function properly, and be able to play songs based on the user's voting.

Future developments

Docker

For easier deployment, creating a docker-compose file would be a great idea, and would make Continuous Integration possible.

Labeling user data for automated suggestions based on AI

With enough collected data a learning algorithm should be able to make recommendations to group of people based on their previous listening history, their likes and dislikes

Keywords

web service, distributed, scalable, Rust, Svelte, MongoDB, nginx, monolithic

References

[1] Creative Commons — CC0 1.0 Universal
URL:https://creativecommons.org/publicdomain/zero/1.0/
(date of access 2022.05.29)
[2] CanIUse
URL:https://caniuse.com/
(date of access 2022.05.29)

[3] Polyfill - MDN Web Docs Glossary: Definitions of Web-related terms

URL:https://developer.mozilla.org/en-US/docs/Glossary/Polyfill

(date of access 2022.05.29)

[4] From JavaScript to React | Learn Next.js

URL:https://nextjs.org/learn/foundations/from-javascript-to-react

(date of access 2022.05.29)

[5] Angular

URL:https://angular.io/

(date of access 2022.05.29)

[6] Virtual DOM and Internals – React

URL:https://reactjs.org/docs/faq-internals.html

(date of access 2022.05.29)

[7] React

URL:https://reactjs.org/

(date of access 2022.05.29)

[8] Stack Overflow Developer Survey 2021

URL:https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks

(date of access 2022.05.29)

[9] Web Archive - BetweenTheWires

URL:https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/

(date of access 2022.05.29)

[10] LLVM

URL:https://llvm.org/

(date of access 2022.05.29)

[11] 25 years of JavaScript history | JetBrains: Developer Tools for Professionals and Teams

URL:https://www.jetbrains.com/lp/javascript-25/#e_2011_03_14

(date of access 2022.05.29)

[12] React vs. Svelte: The War Between Virtual and Real DOM | by Keshav Kumaresan | Bits and Pieces

URL:https://blog.bitsrc.io/react-vs-sveltejs-the-war-between-virtual-and-real-dom-59cbebbab9e9

(date of access 2022.05.29)

[13] NGINX

URL:https://www.nginx.com/

(date of access 2022.05.29)

[14] Welcome to NGINX Wiki!

URL:https://www.nginx.com/resources/wiki/

(date of access 2022.05.29)

[15] sirv - npm

URL:https://www.npmjs.com/package/sirv

(date of access 2022.05.29)

[16] Redis

URL:https://redis.io/

(date of access 2022.05.29)

[17] PostgreSQL

URL:https://www.postgresql.org/

(date of access 2022.05.29)

[18] MongoDB

URL:https://www.mongodb.com/

(date of access 2022.05.29)

[19] PostgreSQL

URL:https://www.postgresql.org/

(date of access 2022.05.29)

[20] PSARAS, Yiannis; DIAS, David. The interplanetary file system and the filecoin network. In: 2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S). IEEE, 2020. p. 80-80.

(date of access 2022.05.29)

[21] Flask

URL:https://flask.palletsprojects.com/en/2.1.x/

(date of access 2022.05.29)

[22] Jinja

URL:https://jinja.palletsprojects.com/en/3.1.x/

(date of access 2022.05.29)

[23] Python.org

URL:https://www.python.org/

(date of access 2022.05.29)

[24] Next.js

URL:https://nextjs.org/

(date of access 2022.05.29)

[25] Node.js

URL:https://nodejs.org/en/

(date of access 2022.05.29)

[26] Deno

URL:https://deno.land

(date of access 2022.05.29)

[27] Rust Programming Language

URL:https://www.rust-lang.org/

(date of access 2022.05.29)

[28] MongoDB Rust Driver

URL:https://www.mongodb.com/docs/drivers/rust/

(date of access 2022.05.29)

[29] Round 20 results - TechEmpower Framework Benchmarks

URL:https://www.techempower.com/benchmarks/

(date of access 2022.05.29)

[30] https://www.jetbrains.com/idea/

URL:https://www.jetbrains.com/idea/

(date of access 2022.05.29)

[31] git-bisect Documentation

URL:https://git-scm.com/docs/git-bisect

(date of access 2022.05.29)

[32] Git

URL:https://git-scm.com/

(date of access 2022.05.29)

[33] Install WSL | Microsoft Docs

URL:https://docs.microsoft.com/en-us/windows/wsl/install

(date of access 2022.05.29)

[34] Rustup

URL:https://rustup.rs/

(date of access 2022.05.29)

[35] GitHub - AlexxNB/tinro: Highly declarative, tiny, dependency free router for Svelte's web applications.
URL:https://github.com/AlexxNB/tinro
(date of access 2022.05.29)

[36] GitHub - AlexxNB/tinro: Highly declarative, tiny, dependency free router for Svelte's web applications.
URL:https://github.com/AlexxNB/tinro#manage-hash-and-query
(date of access 2022.05.29)