Distributed Systems, Challenge Task: Wördle

Team

Team members: Leonardo Ravani, Roger Marty, Tseten Emjee

Responsibilities: - Leonardo: Frontend - Roger: Load balancing, dockerization, database, scheduled task - Tseten: Backend

Project Description

Wordle is a well-known game which gained a lot of popularity in the beginning of 2022. It's a game where you have a five letter word that you need to guess. You get six tries. After every word you type in you get information about your guess. If a letter doesn't occur in the answer it turns grey, if it occurs in the answer but not at that location it turns yellow and if it's also at the right place it turns green.

Inspired by: https://www.nytimes.com/games/wordle/index.html

Our goal was to build this game ourselves for the challenges task. We also wanted to change it up a bit so we implemented the option to choose the word length. One drawback of the "official" Wordle is that you can only play it once per day. Which is why we wanted to implement our own version that you can play as much as you want and also with more words available. The API that we use also has more words available than the "official" Wordle which also makes it a bit harder.

Technologies

- Frontend: HTML, CSS, Javascript, AnimeJS, Bootstrap
- Backend: Express.js
- Loadbalancer: Traefik
- Database: Supabase (open-source Firebase alternative)
- Dockerization: via docker-compose file
- Scheduled Task: Cron for Node.js

Architecture

Frontend

The Frontend is written fully in Javascript, CSS and HTML and to create a sleek and visually appealing design Bootstrap and AnimeJS were utilised. The whole page is a single page application broken down into three parts (login, menu, game) that are hidden or shown depending on the current circumstances. First The login page is displayed which only requires a username. A request is sent to the api to retrieve the login Token which is then stored as a cookie and attached to every further request as a verification header. Next a Request is sent

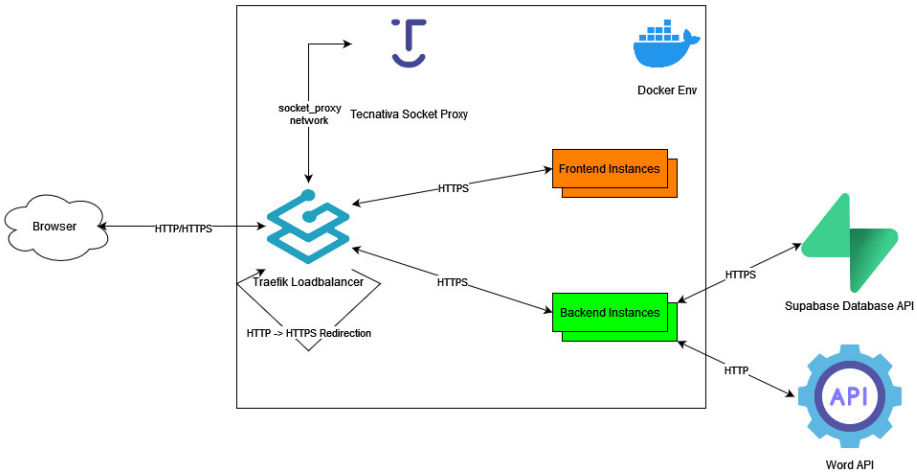

Figure 1: Architecture Diagram

to retrieve the currently open game and the games data (such as the main_word length) is then used to generate the ui of the game page and populate it with the words that have already been guessed. If the data however is empty because there is no open game, the menu page is shown where a user can select the length of the word he wants to guess. After hitting the "New Game" button a request is sent to create a new game with the selected length (default is 5) and the ui of the game page is then generated accordingly and shown. When the API returns that the game is over the current word is compared to the main_word from the response and the game is either lost or won. Errors are visualized using AnimeJS without breaking the game but allowing for the continuation instead.

Backend

Our backend is built using Express.js which we all have previous experience with from the Web Engineering 2 module. The backend is split into three layers to ensure a clean architecture and loose-coupling between classes.

The app.mjs class is our front-facing API Interface Layer, exposing our API routes and handling the request. Second our Business Logic Layer where all the data and game logic gets processed. Lastly our Data Access Layer which handles all the direct calls to our Supabase storage.

app.mjs Express.js setup The setup is quite simple, we have 4 middlewares in total: - API Routes - Page-not-found middleware - Custom error handler middleware - Intermediary JWT middleware for authentication

As you can see before any GET request is processed, we first authenticate the JWT to make sure we have the right user. The token gets signed during the

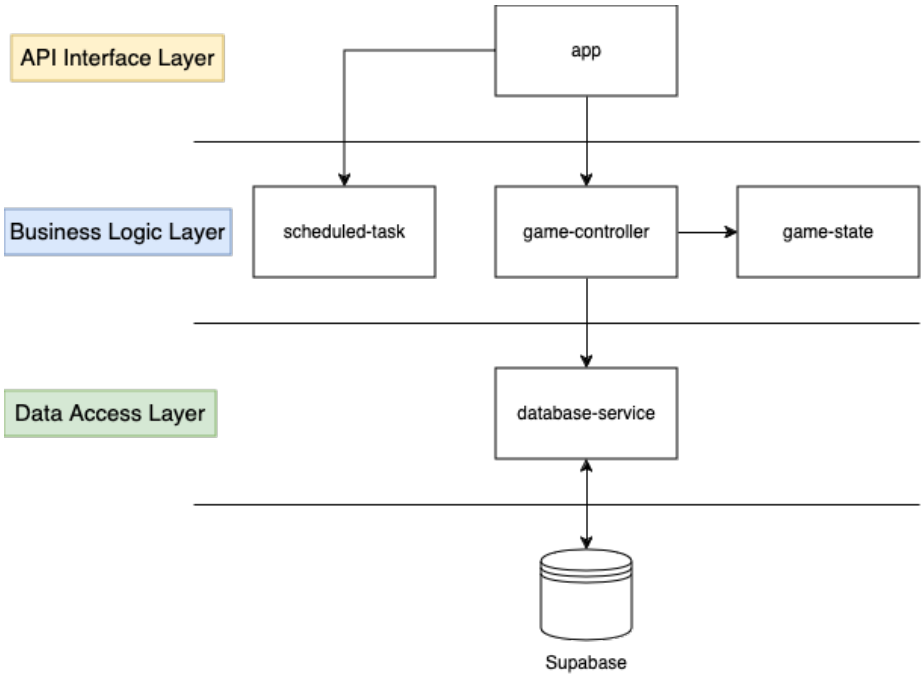

Figure 2: Backend Architecture Diagram

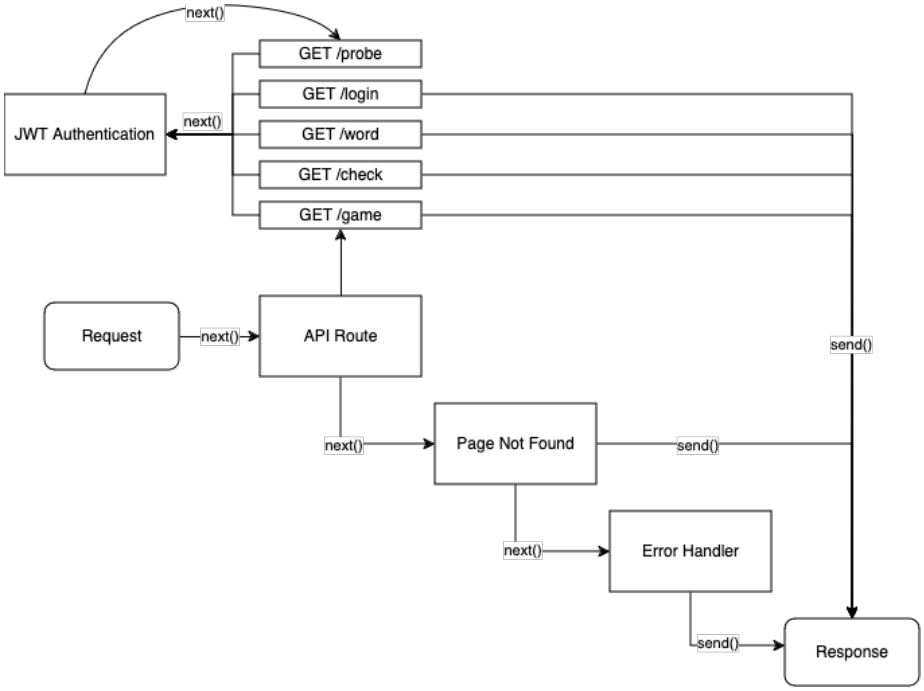

Figure 3: Middleware Diagram

/login call using our token secret and userID. Every other endpoint then requires the token for authentication provided via the Authorization header in the request using the Bearer scheme.

Game Logic The logic for our game is defined in the `game-controller` class. It's straight forward for the most part but the `checkWord()` method should be explained further since it contains the core logic of the application.

The method is called to check if the user provided word matches the hidden one with states reflecting a letters occurrence and placement. For this we use a simple system where the tried words are saved in the database with an additional string reflecting its state. - `g` for green or OK - `y` for yellow or NOK but occurs in hidden word - `x` for error or NOK and doesn't occur

For example let's say the hidden word is `salad` and you try `bangs`, the solution string would be `xgxxy`.

API Endpoints `GET /login/:username`

```
Request:    https://woerdle.localhost/login/test

Response:   HTTP/1.1 200 OK
            X-Powered-By: Express
            Content-Type: application/json; charset=utf-8
            Content-Length: 268
            ETag: W/"10c-+lGpsgQZEzlBVtEsfRnslfGXKDk"
            Date: Thu, 23 May 2024 11:12:51 GMT
            Connection: close

            [{"id":"example-id","username":"test","token":"example-token"}]
```

`GET /word/:length`

```
Request:    https://woerdle.localhost/word/3
Header:     Authorization: Bearer example-token

Response:   HTTP/1.1 200 OK
            X-Powered-By: Express
            Content-Type: application/json; charset=utf-8
            Content-Length: 118
            ETag: W/"76-dL2tHpzSBkQsCr+nZvktX+2JlFE"
            Date: Thu, 23 May 2024 11:15:41 GMT
            Connection: close

            [{"id":143,"main_word":"wye","user_id":"example-id","is_completed":false,"game_s
```

`GET /check/:word`

```
Request:    https://woerdle.localhost/check/lol
```

```
Header:     Authorization: Bearer example-token

Response:   HTTP/1.1 200 OK
            X-Powered-By: Express
            Content-Type: application/json; charset=utf-8
            Content-Length: 167
            ETag: W/"a7-dZg1ERKCaRJeN1KeBHk4vRSu9Dw"
            Date: Thu, 23 May 2024 11:17:16 GMT
            Connection: close

            {"id":143,"main_word":"wye","user_id":"example-id","is_completed":false,"game_st
```

```
GET /game/

Request:    https://woerdle.localhost/game
Header:     Authorization: Bearer example-token

Response:   HTTP/1.1 200 OK
            X-Powered-By: Express
            Content-Type: application/json; charset=utf-8
            Content-Length: 169
            ETag: W/"a9-Wy9ywxvqOO4QelyXZsmszHA5fe8"
            Date: Thu, 23 May 2024 11:19:32 GMT
            Connection: close

            [{"id":143,"main_word":"wye","user_id":"example-id","is_completed":false,"game_s
```

```
GET /probe/

Request:    https://woerdle.localhost/probe

Response:   HTTP/1.1 200 OK
            X-Powered-By: Express
            Content-Type: text/html; charset=utf-8
            Content-Length: 2
            ETag: W/"2-nOO9QiTIwXgNtWtBJezz8kv3SLc"
            Date: Thu, 23 May 2024 11:23:49 GMT
            Connection: close

            OK
```

Loadbalancer

- Traefik used as loadbalancer and entrypoint
- Configured via static and dynamic traefik configuration files
- Tecnativa socket proxy used for extra security when exposing docker socket
- Exposed on ports 80 and 443, HTTP is redirected to HTTPS

- Used together with mkcert as workaround for self-signed certificate warning (see certs folder for script)
- Forwards `woerdle.localhost` requests to frontend and woerdle.localhost/api/... requests to backend (CORS is not an issue)

Failover There are always two frontend and backend instances running to ensure that the service is still available even if one fails. This can easily be scaled up by changing the replica numbers in the docker compose file or implementing this in a Kubernetes environment. The Traefik dynamic configuration contains the configuration for the loadbalancer. It makes a healthcheck to all instances every three seconds and if one doesn't answer for one second it counts as down. All traffic is then redirected to the other instances that are still up. The healthchecks continue in case the node comes up again. The sessions are configured as non-sticky.

Database

For our database we went with a "serverless" approach using the Firebase alternative Supabase. The setup is trivial, merely using Supabase's own JavaScript library and configuring your API key, which is handled by using an .env file, and project URL. The query language is intuitive and simple to use:

```
async loginNewUser(user) {
    return this.supabase
        .from('user')
        .insert([
            { username: user},
        ])
        .select();
}
```

Database Schema The database schema is simple.

We have a `user` table with a username and id. Every User has none-or-many `game-entry` entries, which contains the main word and a boolean field to show if the game is completed, as well as a game state field in JSON format. The game state JSON holds all the tries done by the user:

```
{
  "words": [
    {
      "word": "TESTE",
      "solution": "xyxxx"
    },
    {
      "word": "HELLO",
      "solution": "xyxxx"
```

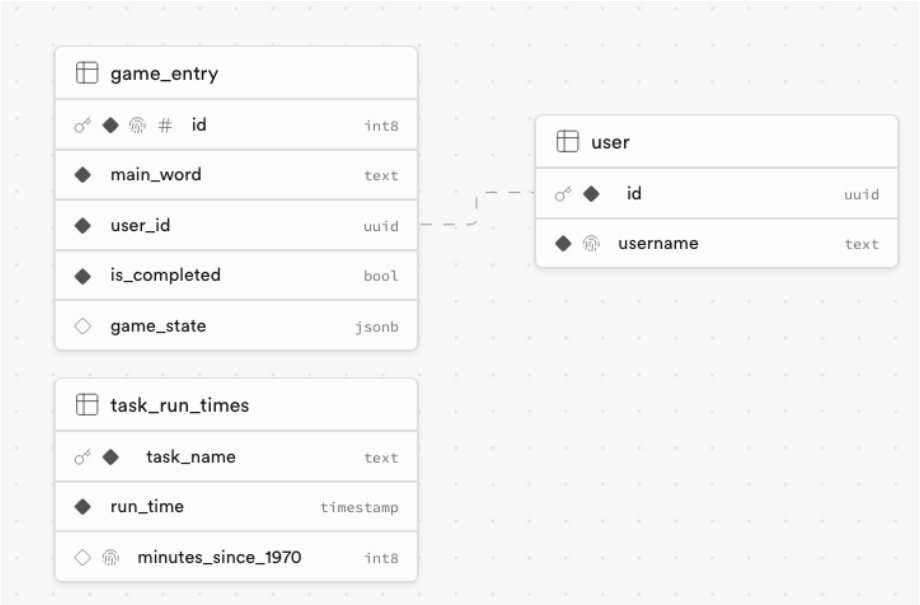

Figure 4: Database Schema

```
    }
  ]
}
```

Scheduled Task

In the Supabase database all game entries are saved. Every game entry also has a boolean column (`is_completed`) which states if a solution has been found and therefore depicts if a game is finished. We configured a scheduled task which runs every minute and checks this table for completed games. These aren't needed anymore and the task deletes them to clean up the database.

When a backend node starts it first makes a new entry in database with the container ID as the name. All backend nodes then use the cron Node.js package for the scheduled task. The task runs every minute on all nodes. To make sure that the task only gets executed once, each node tries to perform an insert operation in the row they created in the beginning. The operation inserts minutes since 1970 in the respective column. This column is UNIQUE meaning that only one backend node succeeds with the task. The node that succeeds is the one that executes the scheduled task. This ensures that it only happens once per minute and also ensures that if one node goes down the task still gets executed because one of the other nodes can now perform the insert operation.

Lessons Learned

Leonardo Ravani

My main objective was to create the frontend and then connect it to the backend using the API calls provided by Tseten. The most interesting part to me that was also discussed the most is how we should organise the API responses and what is saved on the database to ensure that even with one of the backend instances crashing we can still continue the game as it was. I have never worked on a project having to keep in mind that an instance could crash at any given point.

On top of this until now I had only developed textbook frontend pages. This project required a little ingenuity as I wanted to replicate the setting of the already popular wordle game. This required a box for each letter when writing a word as well as the possibility to type and erase the current word while the focus is anywhere on the page. For this the solution was to use eventlisteners on the whole document and calling functions that add, remove classes to organize the divs as well as reading their contents to understand at which point in the word we are currently typing.

To further develop the frontend and make the visual quality match the backend I dove into animation using anime.js for the first time. It is an amazing library that has an extremely helpful and visual documentation showcasing all the possibilities and I will definitely rely on it for all my future projects.

Roger Marty

Even though I already had some experience with Docker I was able to deepen my knowledge. Especially with how the repository should be structured when using a monorepo.

I also already used Traefik a few times but not the load-balancing part of it. I really enjoyed learning more about it and seeing the possibilities Traefik has to offer. It was really nice to configure together with the static/dynamic configuration files (often used labels before). But for upcoming projects I will try some alternatives which interest me aswell e.g. Caddy.

Normally when other projects require a database it will usually be a Postgres container. But for this project we decided to try something different called Supabase. It's an open-source alternative to Firebase. I really enjoyed using it as the UI is really nice to use and it offers many API endpoints. And of course things like availability and scaling are handled automatically. The documantion is also good.

The scheduled task was something that I underestimated at the beginning. I implemented a first version but of course there were some cases that didn't work because I wasn't using transactions. I then searched for some other methods that were possible and Mr. Bocek gave me a possible method that I have never

used. The idea was to make a unique column and all backend instances write the minutes since 1970. Only one instance will be successful and will be the one that executes the task.

Tseten Emjee

I was mainly tasked with building our backend. Though I had some Express.js experience from the Web Engineering 2 module, working on an actual project deepened my understanding of middlewares and in general backend design.

This was also my first time using Supabase. I worked with Firebase before and I have to say for pure storage / database usage Supabase is way easier to use and I can recommend it to anyone looking for a serverless solution. I was most impressed by their API documentation for their client libraries since they generate code specifically for your tables instead of examples.

The biggest lesson I learned though was doing something right is often better than doing something simple. In an early iteration of our application we didn't use JWT authentication and just send user ids around to keep things as simple as possible. While it seemed to work fine initially, later on we discovered that this lead to a number of bugs in our system. Had we just used JWT from the start we would have avoided all that headache.

Usage

Environment Variables

For this project to work two environment variables are needed. In the project root folder there is a `.env` file. Edit this file and replace the example values with the Supabase API key and any desired JWT token secret.

Certificates

If you don't want the self-signed certificate warning when connecting to the Wördle site you can make use of the mkcert tool. This is an open-source tool for creating locally-trusted certificates but only in development environments.

For this a script has been created which creates a certificate and key file for the domain woerdle.localhost. Change directory to the certs folder and execute the script after making it executable. The output shows for which domains a certificate has been created.

```
# make it executable
chmod +x cert-gen.sh

# run the script to generate certificate
./cert-gen.sh
```

Docker Login

To be able to use the docker compose file you need to execute the `docker login` command below. The compose file fetches the newest docker images of our application directly from the GitLab registry. They get built everytime the pipeline runs. A deploy token has been created for this procedure.

```
echo <DEPLOY-TOKEN-PW> | docker login registry.gitlab.ost.ch:45023/roger.marty/woerdle:lates
```

If you don't want to do this or token has expired feel free to uncomment the `build` lines in the docker compose file and comment out the `image` lines. Then the images get directly built from the locally available code.

Docker Compose

After above steps you can finally run the app with below command. Make sure you are in the root directory of the project. This creates all containers defined in the file and sets up everything that is needed.

```
# create environment
docker compose up -d

# check status
docker ps

# tear down environment
docker compose down
```

Connection

The app can then be accessed in the browser by following link: woerdle.localhost

If you want to access the API use following link: woerdle.localhost/api

For access to the Supabase database environment contact one of the team members.