**CCA - Competitive Coding Arena**

**Функционалност:**

Системата предоставя множество функционалности спрямо ролята.

* Потребител - да се регистрира и автентикира в системата.
* Състезател - всеки един състезател има възможността да се записва и участва в състезания, да получава информация относно оптималността и верността на своите решения, да преглежда стари състезания, в които е участвал, както и своите предадени решения.
* Организатор - всеки един организатор има възможността да създава и провежда състезания, да добавя участници, да следи в реално време резултатите на състезателите, да управлява, модифицира и трие състезание
* Администратор - всеки един администратор има достъп до следните функционалности: премахване и промотиране на потребители.

**Архитектура:**

Проектът е разделен на три основни части: фронтенд, бекенд и пуул от докер контейнери под формата на клиент-сървър приложение.

Фронтендът е реализиран под формата на SPA (Single Page Application), който предоставя бързодействие и оптималност откъм заявки към бакенда с цел предоставяне на файлове. Бекендът е реализиран под формата на REST APIs, като се използват и SSE (Server Sent Events) за предаване на информация в реално време към клиента. Бакендът играе роля на медиатор между клиента и докер пуулът, който осигурява изпълнението и оценяването на кода. Докерите представляват среди с поддръжка на Го за сървъра, който управлява изпълнението на кода, и поддръжка на езика, за който е създадено състезанието.

**Реализация:**

Реализацията е развита в няколко стъпки:

* Създаване на база данни и модифициране да работи с реални обекти.
* Поддръжка на докер изпълнение.
* Едновременно и постъпково реализиране на фронт и бек енда.

Основната цел при създаването на системата е възможност за скалируемост и конфигурируемост.

**Модел на данните:**

Не е използвана е релационна база данни и всички обекти се запазват под формата на масиви от байтове. Използва се скоупинг с цел реализация на дейта сетове.

**Конфигурация:**

За конфигуриране се използват .env файлове, които могат да бъдат модифицирани в зависимост от нуждите на съответната система. Различни параметри могат да бъдат задавани като портове, дефолтни потребители, ключ за подписване на JWT.

Системата изисква поне версия 1.21 на GO, версия 3 на Svelte, инсталиран докер и достъп до интернет.

**Използвани технологии и библиотеки:**

* Фронтенд: Svelte със svelte-spa-router, ace.js, sveltestrap и svelte-toast
* Бeкенд: GO, gorilla/mux, godotenv, jwt-go, badger DB, logrus, google/uuid, docker

**Проблеми и решения:**

По време на процеса на създаване възникнаха множество проблеми, които изискваха решения с модерни технологии. Един от тях беше сториджът, който за целите на проекта и курса трябваше да предоставя бързина и оптималност, като скалируемостта беше оставена на заден план. Решението се оказа локална Key-Value система за съхранение на данни - Badger DB. Проблемът тук възниква от това, че за да се поддържат дейта сетове и работа с реални обекти, система се нуждаеше от доработка. Системата поддържа съхранение на масиви от байтове, което не е подходящо за целите на проекта, което доведе до създаването на решение, вдъхновено от Objectify - библиотека за работа с реални инстанции на класове в Java за съхранение в Google Cloud Platform (GCP, Datastore, Firestore). Създадени са интерфейси за работа с реални обекти, които работят с байтове в самата имплементация, а за поддръжката дейта сетове се ползва скоупинг. Друг проблем възникна от нуждата от изпълнение на код в обезопасена и лимитирана среда с цел предотвратяване на атаки, като за целта се използват докери, които са създадени и конфигурирани за определени състезания и поддържат само необходимите функционалности. Третият проблем, който се появи, беше нуждата от SPA (single page application) с цел оптимизиране на работата и улеснено предоставяне на фронтенда от страна на бакенда. В случаят е използван фреймуъркът Svelte. Последният проблем представлява нуждата от уеб-интегрируемо IDE, за което се ползва Ace.js.

**Референции:**

* Идея: <https://www.hackerrank.com/>
* Svelte: <https://svelte.dev/>
* Svelte routing: <https://github.com/ItalyPaleAle/svelte-spa-router>
* GO routing: <https://github.com/gorilla/mux>
* Storage: <https://github.com/hypermodeinc/badger>
* Style: <https://sveltestrap.js.org/?path=/docs/sveltestrap-overview--docs>
* Toast: https://github.com/zerodevx/svelte-toast
* IDE: <https://ace.c9.io/>
* Docker: <https://www.docker.com/>
* Env: <https://github.com/joho/godotenv>
* JWT: <https://github.com/dgrijalva/jwt-go>
* Logging: <https://github.com/sirupsen/logrus>
* UUID: https://github.com/google/uuid