Architecture

Technology picks

Golang

Golang is a highly concurrent and extremely simple language to learn and understand. There is no need to learn complicated frameworks and provides everything one needs to create a web application backend, there are also a lot of language concepts included by default with the language, like the standard library's `net/http` module, and excellent support for modern standards.

The other thing is, I wanted to learn more about the language, as I already have experience with C#, Java, Python, JavaScript when it comes to web development, and this feels like a great opportunity.

Svelte

- Svelte is the framework that best meets the API of the web. It does it'sbest to adhere to standard, and really only builds on what exists, doesn'tinvent anything new like JSX.
- because Svelte follows the web's lead, learning Svelte is super easy. There are no non-standard syntax features, like JSX, and inconsistencies in doing things, like:
  - React functional components
  - React class components
  - old class hooks
  - functional hooks
  - etc.
- The performance is rock solid—Svelte is often beating out other frameworkson UI and JS speed. And it's increasingly used in production on various popular and traction-heavy sites.
- It's the "disappearing framework." This feature is probably Svelte's greatest innovation, and every framework should (and many are trying to) adopt this idea. There are zero client-side dependencies. Just the end JS of the component itself.

Why use svelte in your current project

Code running

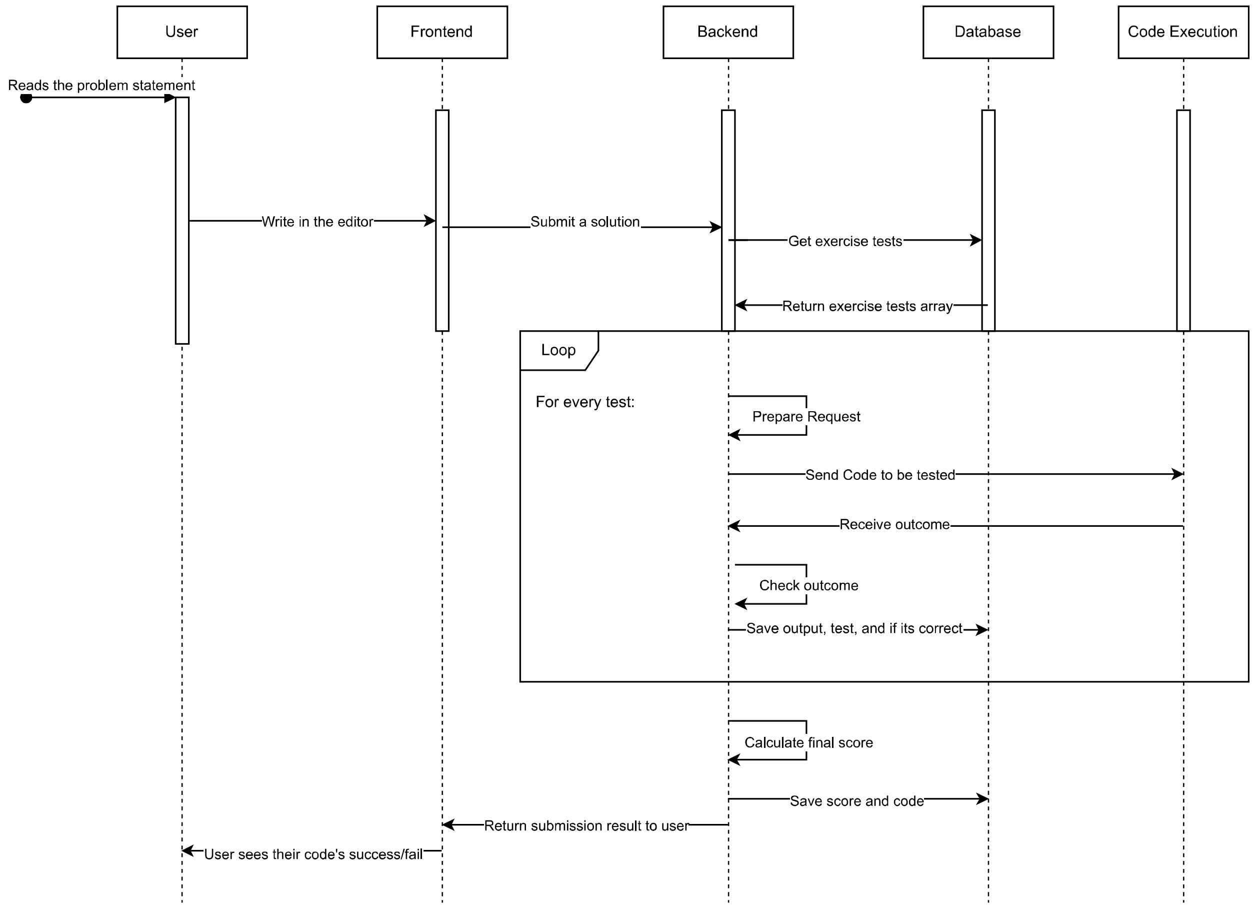
check code-running.md

Business logic

The core part of the application revolves around receiving code from a user, running it with a specific input, and checking the outcome. If the outcome is satisfactory - test passed, otherwise - test failed.

Now, what is the current workflow:

- The application receives a submission that has the user's code.
- The application fetches all the test cases for the given exercise.
- For every test case, a request is sent to the code execution segment, which runs the piece of code, and returns a result to the "backend".
- When a run is complete, the "backend" checks the "output" of that piece of code, and the predefined by that test "output", and if they match, then the test was passed. If there was a runtime error, then the test is automatically failed.
- When all tests have been ran, the results are saved in the database, and returned to the frontend.


User
Frontend
Backend
Database
Code Execution
Reads the problem statement
Write in the editor
Submit a solution
Get exercise tests
Return exercise tests array
Loop
For every test:
Prepare Request
Send Code to be tested
Receive outcome
Check outcome
Save output, test, and if its correct
Calculate final score
Save score and code
Return submission result to user
User sees their code's success/fail