HUNTER
System design document

Authored by
Maninderpal Singh
msc24x@gmail.com

https://thehunter.tech
https://github.com/msc24x/hunter
An open source (GNU General Public Licence) Project
Simplifies daily coding contest hosting

Contents

1. Introduction & Purpose

Competitive programmers are very well familiar with coding contests. There are many popular websites for these competitions. Many already allow users to not only participate, but also to host their own contests.

Hunter in its entirety provides the service to build, manage, host & participate in coding contests in the easiest and simplest way possible. No redundant steps in your way of hosting your own contest. Participants can execute their code on our servers and let our backend Judge evaluate their submissions.

Throughout the competition and after it has ended, Hunter provides a live scoreboard that ranks every participant according to their performance.

In this document, we will be talking about the technical aspect of creating this platform. We will be explaining the tech stack and trade offs made due to security reasons.

Under the top abstraction there are 4 major parts of the app.

1. Authentication
2. Editor
3. Participation
4. Backend Judge

Let's first discuss the tech stack of Hunter.

1.1 Tech Stack Overview

1.1.1 Client

There are many Javascript frameworks to create a web app for our service. For example, React, VueJs, Angular etc. Every framework has its own trade offs and specialities. Even with React being the trending framework, It was decided to build Hunter in Angular. The single page behaviour and a solid development environment of Angular urged this decision. Turns out that Angular worked very well with Hunter.

The project structure consists of key folders-

1. Components

   This contains Angular components that are a small part of the application that are supposed to use in Views.
2. Views

   The Angular components that are logically the pages of the websites. They are generally a bit larger than components.
3. Services

   Angular services that communicate with the REST API and provide basic business use cases for the views.
4. Assets

   This contains images, extra dependency scripts and SCSS colours & mixins.

1.1.2 Server

On the backend side, we have set up an NGINX server that listens on SSL port 443 and serves the static Angular build files of the client app. The server also acts as a proxy for the Hunter's backend API that is an Express server written in TypeScript. Moreover there exists a Docker container that is used to judge users' code submissions through a tester script written in Python. A shell script is used as the mode of communication between the express server and the docker container.

2. Authentication

Before creating an authentication solution for the application we considered different kinds of authentications. Before we explain why the session table was used, let's take a look at our second option, JWT tokens.
Due to enough popularity, JWT tokens drew our attention. So how they work is that the server would require to send clients a cryptic token describing the session that can be decrypted using a public key. The authenticity of that token can be tested by encrypting the contents of the token using the private key and checking if it was not changed. Problem with this was that, what if the token gets stolen due to any reason. Imposters would be able to access the victim's account until the expiration date. And this gets even worse when the server has issued the token for an indefinite duration of time. The only solution to this problem was to maintain a blocklist on our database and check every token against that table. So, this completely destroyed the JWT tokens'

purpose, which was not to touch the database. Hence the process gets equivalent to the session management.

Hence the Hunter uses the session management approach by maintaining a session table in the database, where a session is created for every user login and then the session ID is sent to the client in the cookie. Hence the users are authenticated based upon the session IDs.

So we clearly authenticated an user, but if the user is an actual person is yet to be checked. This is easy as we just need to send a verification email to the intended users. However, due to a technical issue, GitHub OAuth is the only active authentication method being used for now.

2.1 GitHub OAuth

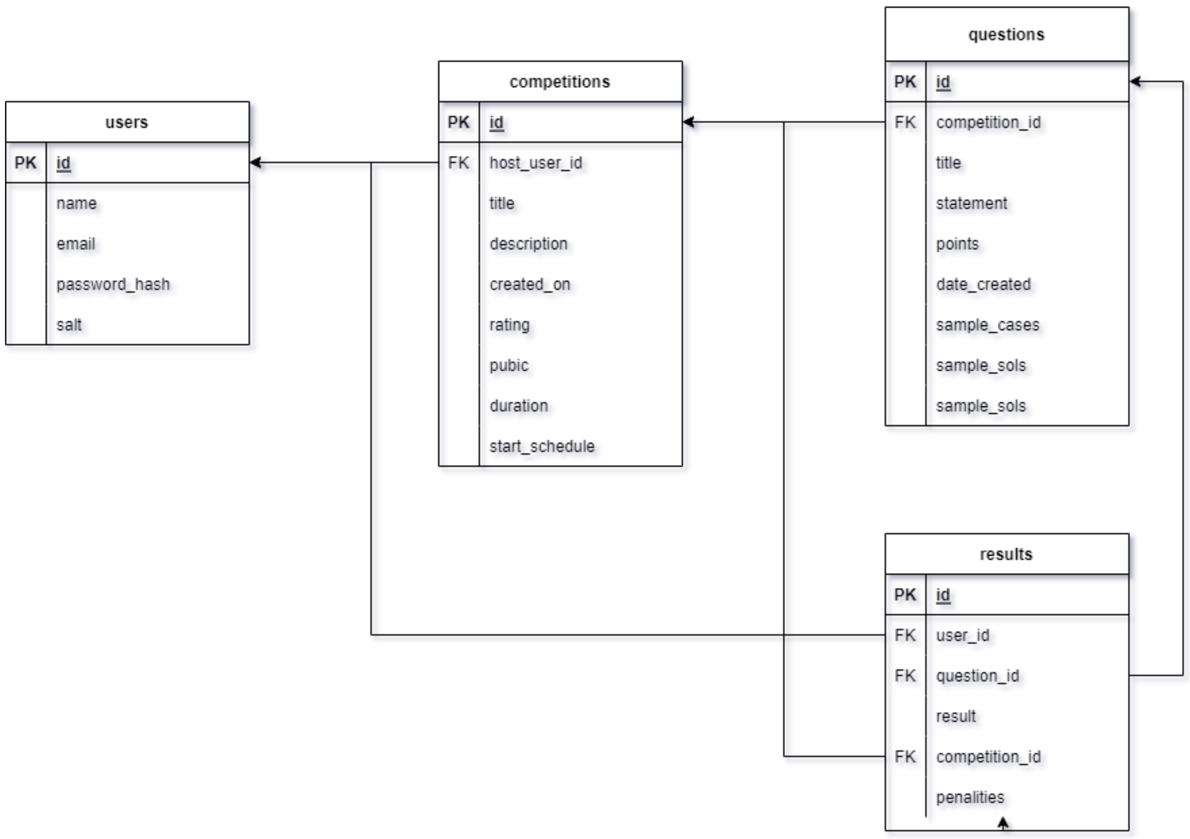
At the time of signing in, you are taken to the GitHub authentication page, where you allow us to access your email id associated. This email is used to verify you as a user for Hunter. We never store your access tokens on our servers.

2.2 Session Management Method

Upon signing in for the first time, a *session* is created for the user *id* in the *sessions table.* This *session id* is stored on your browser *cookies.* Every time you try authentication without passwords, the *session id,* if it exists, is used to verify a valid session record. A logout, invalidates the session id by deleting the record from the database. In this case, you have to log in again.

3. Creating competitions

To allow users to create a competition we needed to deconstruct the data model for it. A contest would consist of its metadata such as its *name, description, host, points, duration, time to go live* etc. One important thing that we encountered was how to store its questions. We thought of creating competition as a parent data structure that would be having its children as questions. But there may be many questions in a competition. One column for all the questions in the competition table would have contradicted the atomicity of the database. So, it was decided to separate the questions. In simple words, there exists a table of all the competitions and another table of all the questions.



The *questions* table contains all the necessary columns for its information and another column for the competition_id that uniquely identifies the competition for what the question is intended for. Hence by doing so any number of questions can be added to a competition. Every time we need to load the questions for the particular competition, we can just search through the questions table with specific competition id.

Now the fetching of a whole competition requires 2 searches in the database, one for the competition and second for the questions.

3.1 Search Optimization

However that can be further optimised by storing the number of questions next to every competition. By doing so we would be able to load the landing page for participation or *editor* quickly. The questions would be searched only when the user clicks on one.
This introduces a new problem. Fetching all the questions requires only one search in the table. But fetching a single question every time a user clicks on one, requires the number of questions times the searches, assuming that every question would surely be visited at least once. That increases disk hits and inner load times at the expense of initial load time.

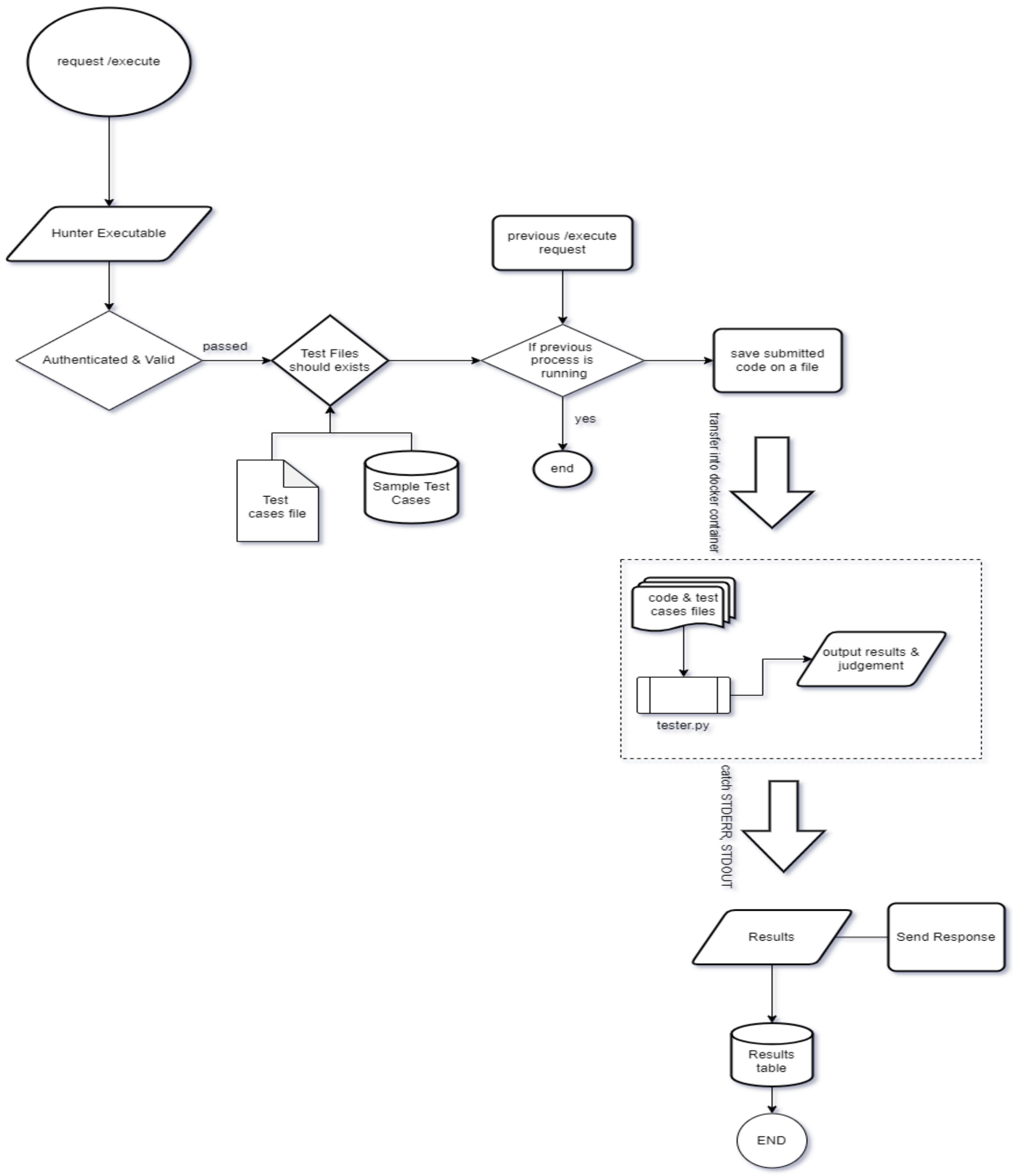
3.2 Timezones

As people using Hunter from different time zones, the *start schedule* was starting to introduce problems by having the type *datetime.* Soon after realising the bug on Azure deployment, we changed the type to VARCHAR, and instead of sending a local time string from the client, it was configured to send UTC time strings to avoid any confusion. The UTC strings are then converted to the local time zone for displaying in the frontend.

4. Participation

To implement the participation, it was necessary to have a code editor, backend pipeline, safe environment for execution and at last, a script to judge the solution and send responses. So for the code editor, Ace editor has been used. It was easy to configure, use and nice theming and highlighting.

4.1 Server Side Code Execution Flow



In Hunter’s terminology, it executes the object called HunterExecutable. This is nothing but a predefined json structure having all the necessary information to execute some code on our servers.

So, this HunterExecutable is taken for authenticated users and is checked against the database if it refers to the actual question of a competition. This is the stage where it is verified if the participants are allowed to submit their code or not. That is, the competition must be live. Now, once we have verified the request to be valid, we do a little pre-check before judging the solution by verifying the existence of test cases data.

The script runTests.sh is the gateway to the docker container where the necessary test cases are copied and the submitted code is passed. The tester.py inside the container acts as the judge. It takes the code, checks the language, compiles it if necessary and executes the code against the input file containing test cases. Any error till this point will be detected and the response will be sent accordingly. The output of the successfully executed code will be compared with the actual solutions to check the correctness of the submission.

It should be noted that the judge prints the result on the console which is captured from outside of the container and is returned in the response payload.

4.2 Files Storage

The submitted code and test cases files are stored on the server in a dedicated directory. The files are given structured names so that we can read/write into it without searching anywhere. This directory also stores the code submitted by users in each language.

4.2.1 Uses

These files are used to serve for the Fetch Last Submission use case for the participants. Since after execution, the submitted code remains there, it can be easily fetched later on using the structured file name.

5. Deployment

Hunter is currently deployed on Microsoft Azure using a Linux VM instance and a MySQL instance. Database *hunter_db* has been deployed on the MySQL instance separately that can be connected only through Azure's resources. That is, the VM can connect to it.

To serve static content and API, an NGINX server has been set up on the VM, that serves the Angular build files and exposes the API through its reverse proxy. Express server is kept running by configuring it as a linux service. The whole application is server over https on the domain thehunter.tech.

6. Conclusion

Hence, we have described why and how the Hunter was built from a high level perspective. The technologies, approaches used while developing may change if required. New features might be added later on. Any update regarding the architecture would be updated on this document from time to time.