Shivansh Yadav

Github: [github.com/Shivansh-yadav13]
Website: *portfolio-shivansh-yadav13.vercel.app*
Resume/CV:
https://docs.google.com/document/d/1VHeCvrpx4Fkolm1GfezMiJnvLkK3o134ff19GcjbN84/edit?usp=sharing
Linkedin: linkedin.com/in/shivansh-yadav-61b619211/
Email: *yadavshivansh@gmail.com*
Phone: *(+91) 9818747879*
TimeZone: GMT +5:30
Location: Delhi, India

Casbin GSoC 2022 Proposal

ABOUT ME

I am a fresher at the Delhi Skill & Entrepreneurship University, India, pursuing my Bachelor of Technology major in Mechanical & Automation Engineering. I have a huge interest in technology & software development. I'm still a beginner & learner, interested in **Web Development** (mostly in backend), **Cloud Native Technologies**.

I got introduced to programming when I was in Grade 11, from there I gained some basic knowledge of **Python** & **MySQL**. After my graduation from school, I started exploring more technologies & languages. I had some basic knowledge of HTML & CSS so I started with Django and Later I switched to React when I was introduced to **Open-Source** & found multiple projects based on **React** or **MERN** Stack.

Since I am new to Open-Source I started by contributing to beginner-friendly organizations & Projects like the **EddieHub Community** & **Community Classroom**. Participated in HacktoberFest for the first and completed the challenge. I am also a student at **GirlScript Summer of Code 22** where we collaborate on various projects created by other student maintainers.

Through these organizations, I learned & gained experience with Web Development & MERN Stack development. I also have an interest in DevOps so I have Knowledge of **Docker, Kubernetes, Golang** and I have developed basic APIs & Web scraper using it.

Casbin is the first big project that I'm contributing to. I have learned many things till now Technically and in Open-Source as well & I'm planning to continue contributing in the future on various different projects.

SKILLS & KNOWLEDGE

Languages - GoLang*(Basic)*, JavaScript/TypeScript, Python*(Basic)*, Java*(Basic)*.
Development - ReactJS*(Intermediate)*, Next*(Basic)*, Node.JS, Express, CSS, TailwindCSS, Bootstrap.
Databases - MongoDB Atlas, MongoDB Realm, MySQL.
Tools - Docker, Kubernetes, Heroku, Vercel, Git & Github.
OS - Windows 10 / Linux Ubuntu 21.

ABSTRACT

As we know Casbin is an authorization library that supports access control models like ACL, RBAC & ABAC. Casbin is extended to multiple languages and frameworks including the Casbin for Node.js.
To improve the user experience for the project one of the factors has to be the performance for which benchmarks may seem necessary to focus on improvement. Another factor in this case can be to stay up-to-date and provide the same features and functionalities like the other Casbin's libraries for eg Casbin's Core Engine written in Go-lang. One more factor to consider can be accessibility over multiple JavaScript platforms which is planned by our mentor Zixuan Liu. To make users utilize Casbin authorization in their work, we can make it consistent with other

latest/most used technologies out there so it becomes easier for users to utilize it.

PROJECT IDEAS

The project I propose to work on for GSoC 2022 is Casbin for Node.js. The aim is to improve the user experience for the Node.js version of Casbin.

I have gone through the project, its codebase and I have decided on a few points that can help us to improve the user experience and Node-Casbin as a whole.

- Implementing Benchmarking & performance enhancement.
- Node-Casin v6 frontend tests and code cleanup.
- Features Enhancement with RBAC w/ Domain API & more with tests.
- Implementation of Apache Cassandra Adapter.

IMPLEMENTATION DETAILS

Implementation of Benchmarks & Performance Improvement

To improve the user experience, benchmarking can help us to track the performance and efficiency of all the existing functions so that we know how and where we can improve. We can implement benchmarking using the Benny package. **Benny** is a simple benchmarking framework that is built on top of the Benchmark package. You can know more about Benny from github.com/caderek/benny.
I have given a small implementation using it below.

```
import { newEnforcer } from '../src'
import { add, complete, cycle, suite } from 'benny';
import { random } from 'lodash';

async function BenchmarkHasPolicy(): Promise<void> {
 const e1 = await newEnforcer('../examples/basic_model.conf', false);
 const e2 = await newEnforcer('../examples/basic_model.conf', false);
 const e3 = await newEnforcer('../examples/basic_model.conf', false);

 for (let i = 0; i < 100; i++) {
   await e1.addPolicy(`user${i}`, `data${i/10}`, 'read');
 }

 for (let i = 0; i < 1000; i++) {
   await e2.addPolicy(`user${i}`, `data${i/10}`, 'read');
 }

 for (let i = 0; i < 10000; i++) {
   await e3.addPolicy(`user${i}`, `data${i/10}`, 'read');
 }

 await suite(
   'BenchmarkHasPolicy',
   add('BenchmarkHasPolicySmall', async () => {
     for (let i = 0; i < 100; i++) {
       await e1.hasPolicy(`user${random(0, 100, false)}`, `data${random(0,
100, false) / 10}`, 'read' )
     }
   }),
   add('BenchmarkHasPolicyMedium', async () => {
     for (let i = 0; i < 100; i++) {
       await e2.hasPolicy(`user${random(0, 1000, false)}`, `data${random(0,
1000, false) / 10}`, 'read' )
     }
   }),
   add('BenchmarkHasPolicyLarge', async () => {
     for (let i = 0; i < 100; i++) {
       await e3.hasPolicy(`user${random(0, 10000, false)}`,
`data${random(0, 10000, false) / 10}`, 'read' )
     }
   }),
   cycle(),
   complete()
   );
}

BenchmarkHasPolicy();
```

We can implement benchmarking just like from the above example for all the other functions and use **ts-node** to run them and get the results (more can be discussed with the mentors).

Output:

```
Running "BenchmarkHasPolicy" suite...
Progress: 100%

  BenchmarkHasPolicySmall:
    924 ops/s, ±2.97%   | fastest

  BenchmarkHasPolicyMedium:
    202 ops/s, ±0.75%   | 78.14% slower

  BenchmarkHasPolicyLarge:
    22 ops/s, ±0.65%    | slowest, 97.62% slower

Finished 3 cases!
  Fastest: BenchmarkHasPolicySmall
  Slowest: BenchmarkHasPolicyLarge
```

We can try going over some APIs to find the best way to boost their performance or making them more user friendly by going over all the possible ways users can be using them and find the efficient method. Similarly the discussion on GitHub was going on for Scaling Access Control Lists for multi-million users as suggested by the author of the issue #147 The implementation can be very useful and can help us to avoid the need for watchers when using an Access control list. We can have a discussion with the mentors for the same if needed and implement it accordingly.

Feature Enhancement with RBAC w/ Domain API & more w/ tests

One more way to improve the user-experience can be to introduce the missing APIs, a few of which are also requested by the users on the documentation website. Node Casbin is not up-to-date with Casbin's Core Engine and there are some features that can be implemented. In the documentation, multiple features present in Go-Casin are not present in Node-Casbin. We can add **RBAC w/ Domain API**, few more related to enforcers like Distributed Enforcer, Enforce with matcher, EnforceEx with Matcher and more missing APIs if not exists.

Some discussions were going on previously on GitHub about adding support for Multiple policies configuration, during the discussion the support was added to the beta version i.e Node-Casbin v6 which is yet to be released, we can add the support for the main branch since the release of the new version's time is not fixed right now.

Implementing a Cassandra Adapter for Node-Casbin

Apache Cassandra as we may already know is a NoSQL database management system designed to handle large amounts of data across many commodity servers is one of the most efficient and **widely-used NoSQL databases** being utilized by multiple companies around the world like Facebook, Netflix, Reddit etc. . We can implement a cassandra adapter and It can be easily utilized by the users using Apache Cassandra in their work. I will work on setting up the entire adapter.

Node-Casbin v6 Frontend tests & Code clean up

Node-Casbin v6 is planned to be supported for all the JavaScript platforms available, we need to add tests for all those JavaScript platforms to ensure it's usability before actually releasing it. Some code clean up for all those dependencies that were removed to support the browser use and some more fixes and implementation changes required.

MILESTONES

Milestone 1

Benchmarks will be added keeping them consistent with the implementation in Casbin Go-lang. Will go through the APIs and will try to boost their performance, will be scaling ACL by the suggested methods during the Github discussion.

Working on the upcoming Node-Casbin v6, cleaning up the code with APIs utilizing “path” or “fs”, rebuild newEnforcer with new params and working on making it useable on all JavaScript platform, testing and adding tests for all the JavaScript platforms like React, React Native, NG, electron etc...

Milestone 2

We will be adding all the missing APIs in Node-Casbin, along with support for multiple policies configuration with their respective tests to keep Node-Casbin up-to-date with other Casbin libraries.

Milestone 3

Working on Implementing Cassandra adapter for Node-Casbin, with both the necessary methods “LoadPolicy()” & “SavePolicy()” and also the optional methods like “AddPolicy()”, “RemovePolicy()”, “RemoveFilteredPolicy()” etc..., We can also add Auto-Saving to have a better performance over “SavePolicy()” we can add all these with reference to the other Node-Casbin Adapters already implemented we will be able to create the adapters and add respective tests and CI to it.

DETAILED TIMELINE

The detailed timeline below contains approx dates but mostly will remain the same, because I’m having my offline college and university semester exams between the timeline. I will be active as much as I can during the time so that I can keep a good conversation between the mentor and me.

May 20 - June 12	**Community bonding period:** discussion with mentors about the idea’s implementation details.
June 13 - June 20	Implementing **Benchmarks** for the node-casbin project for multiple APIs

June 21 - June 30	Working on to improve **Performance** of APIs. **Scaling Access Control List**.
July 1 - July 12	**Node-Casbin v6:** Testing and adding tests for JavaScript platforms like React, React-Native, NG.
July 13 - July 24	**Node-Casbin v6:** Clean up the code & rebuild **newEnforcer** with new params (beta-branch) & making migration documentation. **(Milestone 1)**
July 25 - July 5	**Feature Enhancement:** RBAC w/ Domain API, Distributed Enforcer, EnforceWithMatcher, EnforceExWithMatcher & more with their respective tests. Adding **Multiple policies configuration.** **(Milestone 2)**
August 6 - August 20	Implementing a **Cassandra Adapter** for Node-Casbin project. **(Milestone 3)**
After August 20...	Working on more improvements that could be done on Node-Casbin, working on updating the docs with latest changes. Exploring more Casbin's projects based on Node.js, Golang etc...

CONTRIBUTIONS PRIOR TO GSOC

PR #347 - fix: *matcher result should be boolean or number* for KeyGet2

Due to a raised issue at #332, we found a bug that we get an error "matcher result should be boolean or number" when keyGet2 is used. Since keyGet2 returns a string value, if used alone the matcher result will come out to be a string as well, and the matcher result as a "string" was not handled in the coreEnforcer. I added a case to handle that situation and added a test to confirm it.

PR #349 - fix: 'eval' not detected

Due to a raised issue at #348**,** it was reported that regular expression was not being used correctly and should not include the "g" flag, due to which "eval" in the matcher was not detected every time by the "hasEval" function. I made this PR to solve this issue by adding a separate regular expression since the old regular expression was being utilized in some other function.

PR #341 & #338 - feat: Added GetImplicitUsersForRole & BatchEnforce

BatchEnforce of the Management API & GetImplicitUsersForRole of RBAC API was present in the casbin core engine which is written in Go-lang.
I added the above concerning the already implemented version in go-lang to keep it consistent and their respective tests in Node-Casbin.

PR #56 - fix: Schema p_type to ptype

From a raised issue #55, we realized that Schema used in the Mongoose Adapter was using the field name for ptype as ‘p_type’ which is not consistent with the field name used in the mongodb adapters of other languages like Mongodb Adapter for Go-lang. To keep it consistent everywhere so that even from other languages those policies could be used the Schema was changed from p_type to ptype.

WHY ME?

I’m new to open-source and Casbin is one of the biggest projects that I’ll be working on. I have made some contributions before to the Node-Casbin project due to which I was able to get the knowledge of the code-base. I have some basic knowledge of Go-lang and that helped me to also understand or get some idea about the implementation in the Casbin’s Core Engine and that helped me to be consistent with that in the Node-Casbin project as well.

From my previous experience, I have improved and I will try to provide more valuable contributions and will not remain trivial. When contribtuing to projects in Casbin the mentor/maintainers do help out and give suggestions to improve and so I feel Casbin and its community is a great community to work with and I will learn a lot while contributing.

I’m willing to contribute to the Casbin community for the next 12 weeks+, I would be able to give 25-30 hours at least per week and sometimes more on holidays. I have gone through the GSoC timeline and I feel I will be able to work with the Casbin Community this summer and after GSoC.