دانشگاه صنعتی نوشیروانی بابل
دانشکده برق و کامپیوتر

پایان نامه برای دریافت درجه‌ی کارشناسی
رشته‌ی کامپیوتر

سامانه اجراي كد از راه دور

دانشجو:
محمدعلی علی پناه

استاد راهنما:
دکتر غلامی

شهریور ۱۴۰۲

[*این صفحه خالی است*]


بسم الله الرحمن الرحيم

تقدیم

به مادر ، پدر و پگاه عزیزم که نور مسیر و امید زندگی من اند.

چکیده

پروژه YARCEE یک سامانه اجرای کد از راه دور است که از firecracker برای ساخت ماشین مجازی استفاده می‌کند. درخواست اجرا از طریق HTTP ارسال می‌شود و خروجی نمایش داده می‌شود. معماری این پروژه مایکروسرویس است و شامل چندین سرویس است که از طریق صف با یکدیگر در ارتباط هستند. در نتیجه اجرای کد آسنکرون است و چندین ماشین مجازی می‌توانند همزمان اجرا شوند. این پروژه شامل رابط کاربری است که در آن می‌توان پروژه جدید ساخت و کد را ویرایش و خروجی را مشاهده کرد.

کلمات کلیدی: اجرای کد ؛ گولنگ ؛ ماشین مجازی ؛ سامانه اجرای کد از راه دور

[*این صفحه خالی است*]

فهرست مطالب

چکیده **الف**

۱ آشنایی با پروژه **۱**

۲ مفاهیم پایه **۳**

۲-۱ Firecracker ۳

۲-۲ Fiber Go ۴

۲-۳ RabbitMQ ۴

۲-۴ PostgreSQL ۴

۲-۵ Next.js ۴

۳ معماری پروژه **۵**

۳-۱ API ۷

۳-۲ VMVisor ۸

۳-۳ Frontline ۹

۴ ویژگی از دید کاربر **۱۱**

۴-۱ کلاینت ۱۱

۵ نتیجه گیری و کارهای آتی **۱۳**

[*این صفحه خالی است*]

فهرست شکل‌ها

۱-۱ لوگو پروژه ۱
۱-۲ ساختار کلی پروژه ۲
۱-۲ نمایی از ساختار Firecracker ۳
۱-۳ معماری پروژه ۵
۲-۳ لیست API ۷
۳-۳ وضعیت اجرا کد ۸
۱-۴ طرح برخی از صفحه ها ۱۲
۲-۴ لیست پروژه ها ۱۲

[*این صفحه خالی است*]

فصل ۱

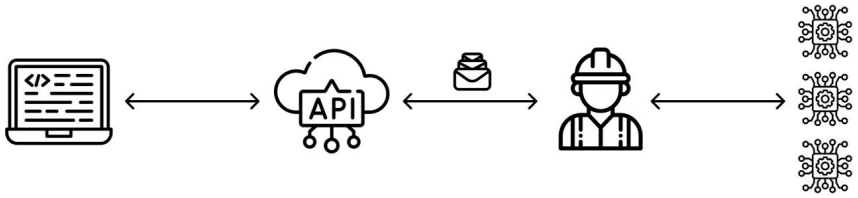
آشنایی با پروژه

شکل ۱-۱: لوگو پروژه

YARCEE مخفف Yet Another Remote Code Execution Engine یک سرویس ابری اجرای کد است. معماری این پروژه مایکروسرویس است و سرویس ها با زبان Go نوشته شده اند. برخی از ویژگی های کلیدی پروژه شامل:

- اجرای کد زبان های مختلف مانند python - nodejs - c - c++ و ...
- عدم دسترسی پروژه ها به یکدیگر به دلیل اجرا در micro-vm
- ادیتور انلاین و ویرایش در مرورگر
- قابلیت ساخت ، ادامه و تغییر نام پروژه
- همروند بودن اجرای کد توسط صف پیام RabbitMQ

این پروژه یک سرویس ابری است که شامل ویژگی های یک محصول واقعی است. کاربر امکان ثبت نام و ورود دارد. می‌تواند پروژه جدید بسازد و آن را ویرایش و حذف بکند. در فصل جمع بندی نگاهی به آینده این محصول می‌کنیم اما قبل از آن به صورت خیلی خلاصه نحوه عملکرد این سیستم را بررسی می‌کنیم.



شکل ۱-۲: ساختار کلی پروژه

پس از ثبت نام و ورود به داشبورد، از قالب های از پیش تعریف شده یکی را به دلخواه انتخاب می‌کنیم. هر قالب شامل اسمی تصادفی، کد اولیه زبان و اطلاعاتی در مورد اجرای آن است. پس از ویرایش کد آن را اجرا می‌کنیم.

در پشت صحنه درخواستی برای دریافت خروجی کد به سرویس API زده می‌شود. وظیفه این سرویس ارتباط مستقیم با پایگاه داده و گوش دادن به وضعیت اجرای کد و خروجی آن است.

این درخواست روی صف RabbitMQ ارسال می‌شود و توسط سرویسی از صف دریافت می‌شود. این سرویس سپس درخواست را برای پردازش یکی از vm هایی که در حالت آماده باش قرار دارد می‌فرستد و خروجی اجرای کد را دریافت می‌کند.

مجموعه از vm ها در حالت آماده باش قرار دارند که پس از اجرای کد حذف و سپس جایگزین به مجموعه اضافه می‌شود. متصل به هر vm فایل سیستمی از پیش ساخته شده که است که باینری کامپایلرهای مختلف درون آن وجود دارد.

وابستگی اصلی این پروژه به Firecracker است که برای مدیریت micro-vm ها به کار می‌رود. برای ارتباط با این ابزار از sdk زبان GO آن استفاده شده است. از مزیت های استفاده از روش micro-vm می‌توان سرعت بسیار بالاتر نسبت به vm اشاره کرد. همچنین این روش امنیت بالاتری نسبت به اجرای کد در محیط های کانتیری مانند docker دارد زیرا micro-vm ها به طور کل از یکدیگر جدا هستند.

بخش مهم دیگر این پروژه محیط وب و ادیتور انلاین آن است. کلاینت با زبان TypeScript نوشته شده است و از فریمورک Next.js استفاده می‌کند. کتابخانه ui زبان به طبع React است. در بخش ادیتور امکان ویرایش کن و درخواست اجرا وجود دارد. با فشردن اجرا کلاینت هر از ۱۰۰ میلی ثانیه به سرور درخواست میزد تا خروجی کد درخواست شده را دریافت کند. پس دریافت stdout و stderr این چرخه را متوقف می‌کند.

در ادامه به معرفی هر سرویس خواهیم پرداخت.

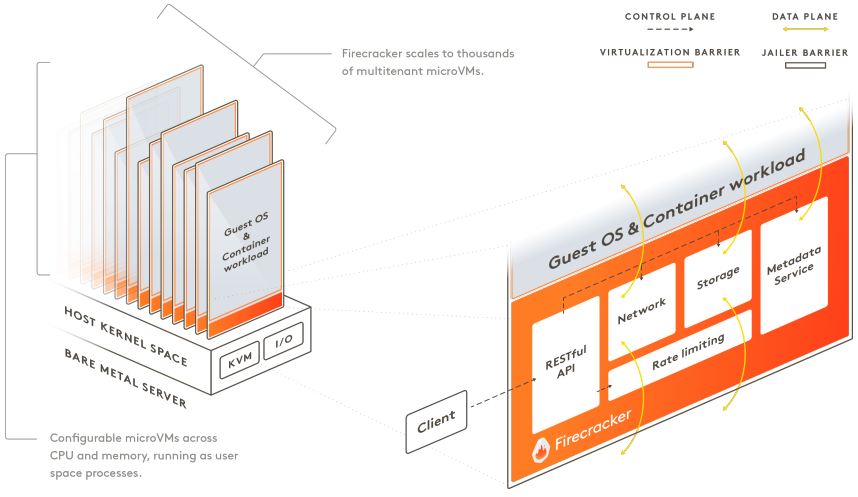
فصل ۲

مفاهیم پایه

در این فصل به معرفی تکنولوژی و فریمورک های به کار رفته در پروژه میپردازیم.

Firecracker ۱-۲

در قلب پروژه Firecracker قرار دارد. Firecracker یک مانیتور ماشین مجازی است که از KVM استفاده می‌کند و وظیفه اش ساخت و مدیریت ماشین های مجازی است. Firecracker توسط تیم وب سرویس آمازون توسعه داده شده و در پروژه Fargate و Lambda این شرکت نیز استفاده شده است.



شکل ۲-۱: نمایی از ساختار Firecracker

Fiber Go ۲-۲

زبان استفاده شده در میکروسرویس ها Go است که توسط گوگل توسعه داده شده و از فریمورک Fiber استفاده شده که برای ساده تر شدن routing و middleware استفاده شده است. از دلایل استفاده از Go میتوان به سادگی و سرعت بالا اشاره کرد. همچنین این زبان در مسائل concurrency ابزارهای low-level زیادی در دسترس کاربر قرار می‌دهد.

RabbitMQ ۳-۲

RabbitMQ یک نرم افزار برای انتقال پیام بین سیستم ها است. در این پروژه درخواست اجرا کد در صف وارد می‌شود و توسط سرویسی پردازش می‌شود. دلیل استفاده از event-driven بلاک نشدن درخواست ها است.

مزیت استفاده از RabbitMQ آسنکرون شدن سیستم است. پیام ها و وضعیت اجرای کد پشت یکدیگر بلاک نمی‌شوند. همچنین سیستم ها از وجود یکدیگر بی خبر هستند و وابستگی شان بهم کم می‌شود. به این معماری loosely-coupled می‌گویند.

PostgreSQL ۴-۲

دیتابیس اصلی استفاده شده PostgreSQL است که از نوع رابطه ای است. جداول این نرم افزار شامل کاربران و کدها است. دلیل انتخاب PostgreSQL مورد اطمینان بودن و سادگی این پایگاه داده بوده است. برای ارتباط و کوئری زدن از کتابخانه gorm زبان Go استفاده شده که ORM محبوبی است.

Next.js ۵-۲

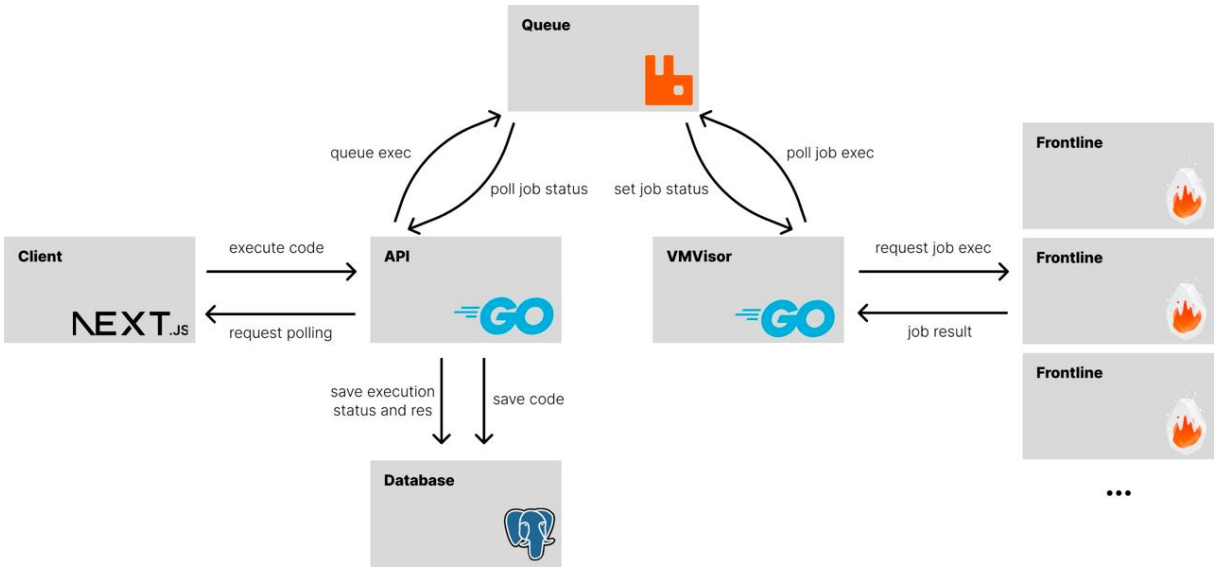
فریمورک استفاده شده سمت کلاینت Next.js است که از کتابخانه React برای رندر روی مرورگر استفاده میکند. دلیل استفاده از React ساده کردن پیاده سازی رابط کابری توسط هوک ها و کامپوننت محور بودن آن است.

زبان برنامه نویسی سمت کلاینت TypeScript است که به واسطه کامپایلر تبدیل به JavaScript می‌شود. دلیل استفاده از TypeScript اضافه شدن شی‌گرایی و تایپ در زمان کامپایل است.

فصل ۳

معماری پروژه

همانطور که در ۳-۱ می‌بینید این پروژه از معماری میکروسرویس بهره می‌برد. در این فصل ابتدا به معرفی کلی هر سرویس و سپس نگاهی دقیق به عملکرد هر سرویس می‌اندازیم.



شکل ۳-۱: معماری پروژه

همانطور که در شکل و جدول مشاهده می‌کنید پروژه از ۴ سرویس اصلی تشکیل شده است. درخواست از کلاینت شروع شده پس از طی کردن API و VMVisor به Frontline می‌رسد. پیش تر اشاره کردیم که Frontline یه API Web است که درون vm در حال اجراست. Frontline در اصل یک API Web است که با پروسه فرزند کامپایلر زبان های مختلف را فراخوانی کرده و خروجی را بر می‌گرداند.

جدول ۳-۱: لیست سرویس ها

سرویس ها	توضیحات
API	سرویس API REST است که وظیفه صحبت با دیتابیس و پاسخ به کلاینت را دارد
VMVisor	مدیریت VM های ساخته و دریافت و تغییر وضعیت درخواست های اجرا روی صف
Frontline	درون هر VM در حال اجراست و توسط پروسه فرزند کامپایلر زبان را صدا می‌زند
Client	کلاینت وطیفه نمایش رابط کاربری و ادیتور را دارد

مهم ترین سرویس این پروژه VMVisor است که وظیفه مدیریت vm ها را بر عهده دارد. این پروژه از Firecracker استفاده می‌کند و مجموعه ای از vm ها را مدیریت و به آن ها وظیفه ای برای اجرا می‌سپارد.

ارتباط بین API و VMVisor به صورت آستکرون توسط انتقال پیام در صف است. در RabbitMQ دو صف وجود دارد. صف درخواست اجرا و صف وضعیت اجرا. در صف درخواست اجرا سرویس API کدی که کلاینت ارسال کرده را روی صف قرار می‌دهد و VMVisor این درخواست را از صف برمی‌دارد. صف دیگر وضعیت اجرا است که VMVisor خروجی Frontline را روی صف قرار می‌دهد و API به آن گوش می‌دهد و روی پایگاه داده می‌نویسد.

وظیفه API عملیات های CRUD(create read update delete) است و اولین درگاهی است که کلاینت با آن در ارتباط است. همچنین وظیفه قرار دادن درخواست اجرا در صف و گوش دادن به صف وضعیت اجرا و نوشتن آن روی پایگاه داده را برعهده دارد. از دیگر وظیفه های این سرویس مدیریت کاربران و پروژه های آن ها است. این سرویس خود پتانسیل شکسته شدن به میکروسرویس های کوچک تر را دارد ولی در این پروژه این تصمیم گرفته نشده است.

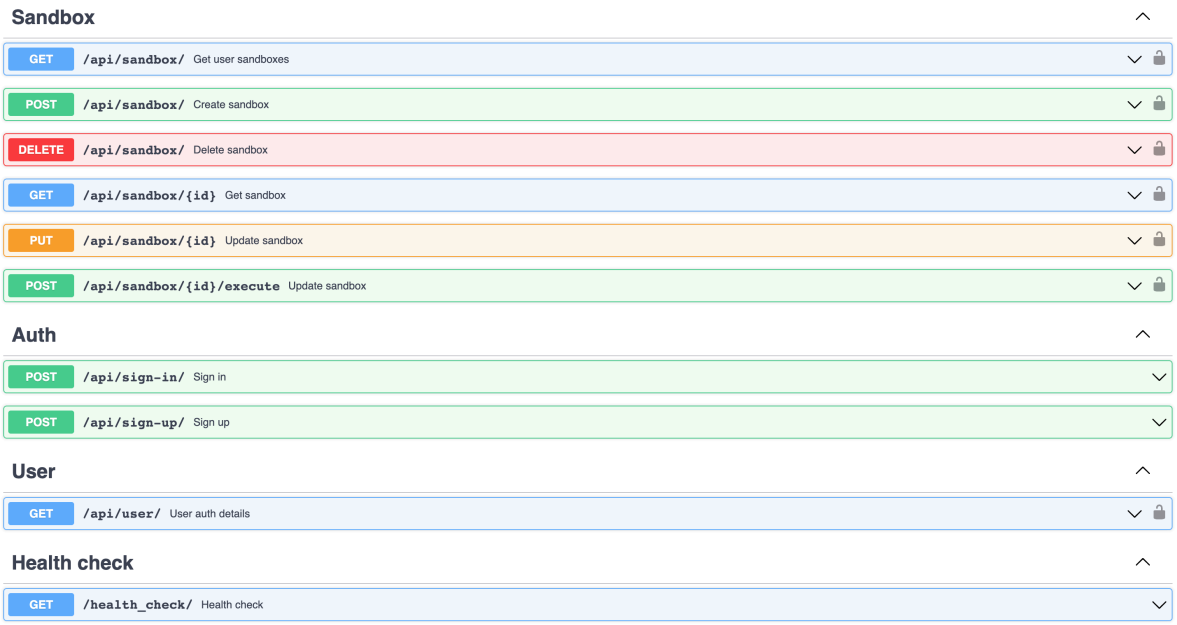
کلاینت هم بخش مهم دیگری است که رابط کاربری سیستم با نرم افزار است. کاربر امکان ساخت پروژه جدید و ویرایش آن در مرورگر را دارد. رابط کاربری این پروژه در Figma طراحی شده و توسط کتابخانه React دیزاین سیستم طراحی شده و کامپوننت های مختلف در کنار هم قرار گرفته اند.

در ادامه به معرفی دقیق تر هر سرویس و ارتباطش با سایرین می‌پردازیم.

API ۱-۳

همانطور که اشاره کردم بخش API پروژه از زبان Go و فریمورک Fiber استفاده می‌کند. این پروژه لایه ورودی ما به بخش های داخلی سیستم است.

در عکس زیر نمایی از api های موجود در پروژه مشاهده می‌شود. این api ها در چند دسته مختلف تقسیم بندی شده اند. sandbox برای ساخت یک پروژه جدید و اجرا آن است. بخش auth برای ثبت نام و ورود کاربر است. بخش user برای دریافت اطلاعات کاربر وارد شده است. و health برای بررسی liveness و readiness سیستم در نظر گرفته شده است. وجود این مسیر باعث می‌شود در سیستم های مدیریت کانتینر مانند kubernetes از آمادگی سرویس اطمینان حاصل کرد.



شکل ۳-۲: لیست API

همچنین این سرویس به دو صف متصل است که به آن گوش می‌دهد و روی آن ارسال می‌کند. کلاینت پس از ساخت پروژه و ویرایش کد، درخواست اجرا آن را ثبت می‌کند. در پشت صحنه درخواستی به مسیر execute زده می‌شود. این درخواست روی صف sandbox queue ارسال می‌شود.

از طرفی سرویس VMVisor این درخواست را به یک vm می‌سپارد و خروجی کد و وضعیت اجرای آن را روی صف sandbox status queue قرار می‌دهد. سرویس API به آن صف گوش می‌دهد و آن را روی پایگاه داده می‌نویسد. کلاینت هر از ۱۰۰ میلی ثانیه در حال درخواست برای stdout و stderr است.

VMVisor ۲-۳

شاید پیچیده ترین سرویس این پروژه VMVisor باشد. این سرویس چندین وظیفه دارد شامل:

- مدیریت و ساختن مجموعه vm
- دریافت درخواست اجرا کد از صف
- ارتباط با Frontline برای اجرای کد
- ارسال تغییر وضعیت کد به صف

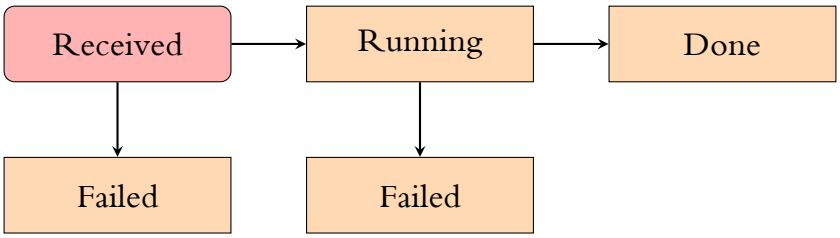
تعداد مجموعه vm های آماده ۱۰ عدد است. به هر vm یک عدد هسته پردازنده و ۲۵۶ مگابایت حافظه RAM و ۱ گیگابایت فضا اختصاص می‌شود.

ماکسیمم مدت زمان اجرا ۱۰ ثانیه است. پس از آن vm حذف می‌شود. پس از اجرا یک vm به مجموعه اضافه می‌شود تا تعداد vm های آماده همان ۱۰ عدد بماند.

ارتباط VMVisor با vm از طریق سرویس به نام Frontline است. Frontline در اصل یک REST API است که درون vm در حال اجراست و درخواست را دریافت کرده و خروجی را به VMVisor بر می‌گرداند. این سرویس سپس وضعیت اجرا را روی صف قرار می‌دهد.

در شکل زیر وضعیت های مختلف اجرا کد را می‌توانید ببینید.

شکل ۳-۳: وضعیت اجرا کد



در ادامه به آخرین تکه پازل یعنی Frontline می‌پردازیم.

۳-۳ Frontline

همانطور که اشاره کردیم Frontline یک Web API است که با زبان Go توسعه داده شده. این سرویس درون vm در حال اجراست و از طریق پردازه فرزند کامپایلر زبان مورد نظر را فراخوانی می‌کند.

این سرویس از طریق openrc که سرویس init برای لینوکس است بلافاصله بعد از بوت vm در حال اجرا قرار می‌گیرد.

درون فایل سیستم هر vm تمام کامپایلرهای مورد نیاز قرار دارد. این فایل سیستم با دستور dd با فضای یک گیگابایت ساخته شده است و از طریق docker روی آن نوشته می‌شود.

این پروسه توسط اسکریپتی انجام می‌شود. در این اسکریپت توسط alpine و پکیج منجر آن کامپایلر زبان های مختلف دانلود می‌شود. فایل سیستم به عنوان volume برای docker اضافه می‌شود و از این طریق امکان نصب این کامپایلر ها روی آن امکان پذیر می‌شود.

هر vm نیاز به گرفتن ip دارد. VMVisor از طریق درخواست HTTP با Frontline در ارتباط است. برای بحث نتورکینگ از CNI استفاده شده است که از بحث این مقاله خارج است.

درون Frontline دو مسیر اصلی وجود دارد. health و .health exec برای بررسی آمادگی vm به کار می‌رود. پیش تر اشاره کردیم که وظیفه VMVisor آماده نگه داشتن ۱۰ vm است. ساختن vm جدید ممکن است چندین ثانیه طول بکشید. هر ۱۰ ثانیه VMVisor درخواستی به health می‌زند و پس از دریافت ۲۰۰ آن vm را به مجموعه اضافه می‌کند.

[*این صفحه خالی است*]

فصل ۴

ویژگی از دید کاربر

۴-۱ کلاینت

همانطور که اشاره کردیم کلاینت این پروژه از Next.js استفاده می‌کند. طراحی رابط کاربری در محیط Figma انجام شده است.

مراحل پیاده سازی به شرح زیر است:

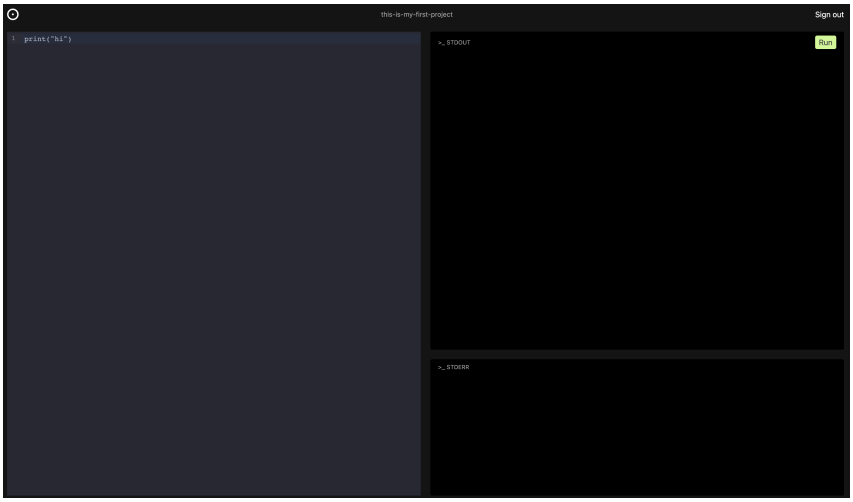
۱. دیزاین توکن ها را استخراج و به پروژه اضافه می‌کنیم. مانند رنگ ها، فاصله ها و سایه ها

۲. المنت های دیزاین سیستم رو پیاده سازی می‌کنیم. کامپوننت هایی مانند دکمه، اینپوت و کانتینر

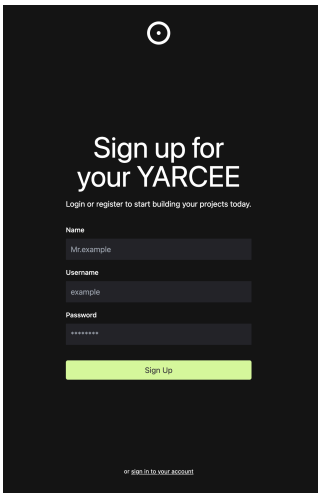
۳. با کنار هم قرار دادن المنت های دیزاین سیستم و کامپونتت های مخصوص هر بخش، صفحه را تکمیل می‌کنیم

۴. مراحل دریافت یا فرستان اطلاعات را انجام می‌دهیم

۵. مرحله ۳ و ۴ را برای هر صفحه تکرار میکنیم



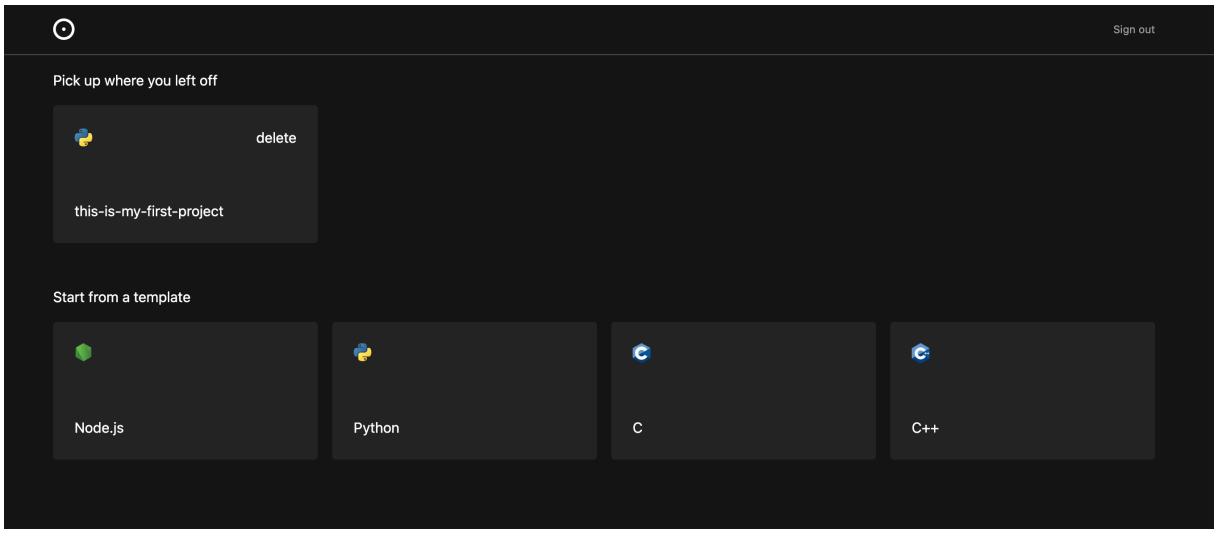
(ب) صفحه ادیت کد



(الف) صفحه ثبت نام

شکل ۴-۱: طرح برخی از صفحه ها

در شکل بالا میتوانید نمونه ای از صفحه های پیاده سازی شده در این پروژه را مشاهده کنید.



شکل ۴-۲: لیست پروژه ها

پس ثبت نام و ورود به سایت شما با صفحه داشبورد مواجه می‌شوید. در این صفحه می‌توانید به ادامه ویرایش پروژه قبلی خود بپردازید یا توسط قالب های از پیش تعیین شده پروژه جدیدی شروع کنید. اضافه کردن اکثر زبان های برنامه نویسی ممکن است ولی در حال حاضر از Node.js، Python، C و C++ پشتیبانی می‌شود.

فصل ۵

نتیجه گیری و کارهای آتی

نتیجه انجام این پروژه برای من یادگیری بهتر زبان Go و آشنایی با firecracker بود. چالش های زیادی رو پشت سر گذاشتم تا به یک نسخه اولیه رسیدم ولی هنوز هم کارهای زیادی مانده تا به یک محصول واقعی تبدیل شود.
کارهای أتی:

- امکان scale افقی. در حال حاضر تنها ۱۰ vm در مجموعه آماده به کار است
- ایجاد فایل و فولدر. در حال حاضر تمام کد باید در یک فایل نوشته شود
- دسترسی به ترمینال
- تغییر کد به صورت گروهی
- قابلیت اشتراک گذاری پروژه به صورت عمومی
- قابلیت اشتراک گذاری به کاربری خاص همراه با دسترسی به ادیت یا قفط مشاهده
- قابلیت fork
- قابلیت ساخت قالب از روی پروژه
- اضافه کردن زبان های بیش تر

[*این صفحه خالی است*]

Abstract

YARCEE(Yet Another Remote Code Execution Engine) is a code running service that relies on Firecracker to spawn microVMs and execute the code over HTTP.

Keyword: Remote code execution - MicroVM - VMM

Babol Noshirvani University of Technology
Faculty of Computer Science

A Thesis
Submitted in Partial Fulfillment of the Requirement for the Bachelor Degree of Science in
Computer

Remote code execution engine

by:
Mohammad ali Ali panah

Supervisor:
Dr.Gholami

August, 2023