JONATHAN CHIASSON

Technologies émergentes

420-5B5-V gr. 00

DOSSIER D'ANALYSE

Projet de session - API en Rust

Travail présenté à

Étienne Rivard

Département des Techniques de l'informatique

Cégep de Victoriaville

Le 16 décembre 2022

CONCEPTS

Voici les concepts qui seront utilisés dans ce document d'analyse et leur définition:

- **API**: *Application Programming Interface*, c'est le terme qui définit le système informatique qui fait le lien entre la base de données et un système donné à l'aide de requêtes.
- **Rust**: Langage de programmation orienté objet développé par Mozilla depuis 2010.
- **MongoDB**: Type de base de données fonctionnant avec des documents en JSON.
- **MongoDB Atlas**: Service d'hébergement de base de données MongoDB.
- **Framework**: En programmation informatique, un framework (appelé aussi infrastructure logicielle, infrastructure de développement, environnement de développement, socle d'applications, cadre d'applications ou cadriciel) est un ensemble cohérent de composants logiciels structurels qui sert à créer les fondations ainsi que les grandes lignes de tout ou partie d'un logiciel, c'est-à-dire une architecture. (Wikipédia)
- **Repository**: Partie de l'API qui s'occupe de l'acquisition des données (dans ce cas-ci, dans la base de données)
- **Service**: Partie de l'API qui prend en charge la logique des requêtes et les transmet au repository.
- **Route**: Partie de l'API qui transmet les requêtes au bon service. Dans le cas du framework utilisé, les parties Route et Service sont mises en une seule partie.
- **Visual Studio Code**: Logiciel de programmation utilisé dans le cadre du projet
- **TypeScript**: Langage de programmation fortement typé basé sur le JavaScript
- **C**: Langage de programmation typé non orienté objet créé dans les années 1970.

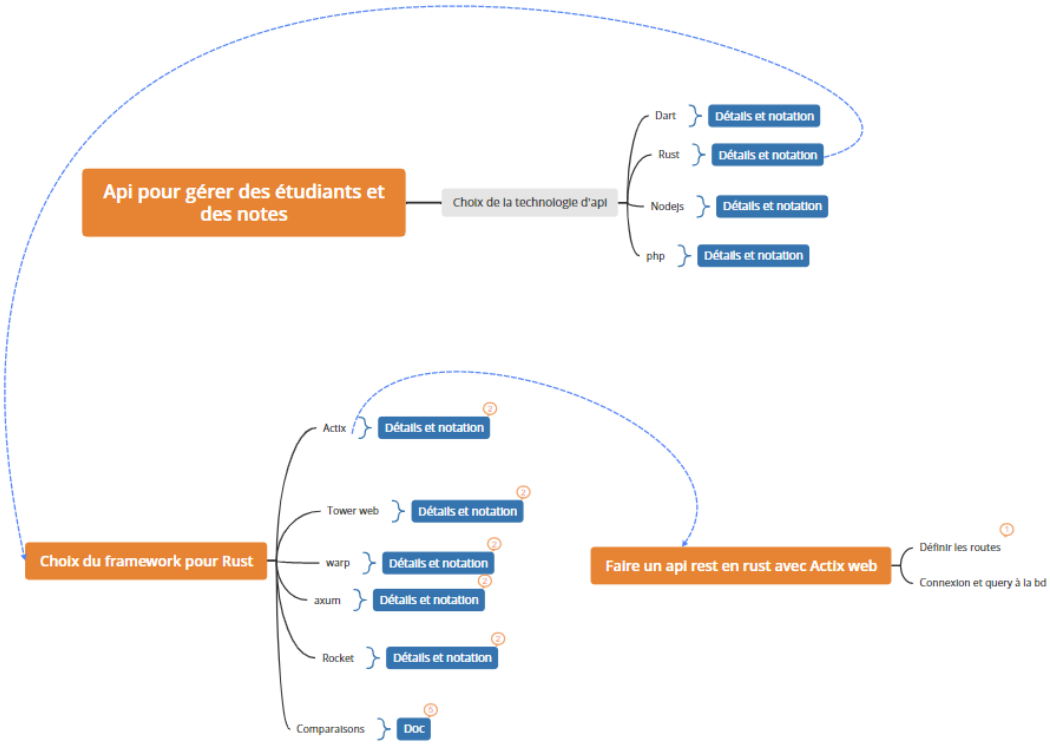
BUT DU PROJET

Le but du projet est de créer un API en Rust qui gèrera des étudiants, des évaluations et des notes.

L'API permettra de créer, modifier et supprimer des étudiants et des évaluations. Il permettra aussi de donner une note aux étudiants à chaque évaluation.

Le système utilisera une base de données MongoDB hébergée sur MongoDB Atlas. Cela permettra d'avoir une idée des performances en situation réelles et facilitera le transfert du prototype entre plusieurs programmeurs.

CARTOGRAPHIE DES CONNAISSANCES



Ma cartographie des connaissances représente le parcours que j'ai eu en cherchant des informations sur les API en Rust.
Elle a été conçue sur GitMind. Elle est disponible en allant sur le lien suivant:

[Cartographie des connaissances - GitMind]

TABLEAU COMPARATIF

Les raisons pour lesquelles j'ai choisi le Rust sont:

- Je ne connais pas ce langage
- Il est réputé comme le langage le plus rapide en la matière

Pour le tableau comparatif, j'ai décidé de comparer les frameworks/bibliothèques disponibles pour faire un api en rust. Je les ai notés selon les critères suivants:

/	Documentation claire	Facilité d'utilisation	Utilisation par d'autres utilisateurs	Présence de bugs	Rapidité	Total
Poids	30	30	10	20	10	100

Le résultat de mes recherches et de ma notation est le suivant:

Technologies	Documentation claire	Facilité d'utilisation	Utilisation par d'autres utilisateurs	Présence de bugs	Rapidité	Total
Actix-web	100	90	90	25	80	79
axum	75	70	50	100	90	77,5
Rocket	100	90	100	30	0	73
warp	75	70	30	0	80	54,5
Tower web	75	30	20	30	90	48,5

J'ai donc décidé d'utiliser Actix-web pour mon api.

JOURNAL DES ENJEUX ET SOLUTIONS

Prorblème	Description	Solution

Le Rust ne permet pas de lier 2 repository	Dans la manière dont le rust démarre, Il déclare le repository au début pour pouvoir le lier automatiquement à ses services (routes). Cette manière de fonctionner de mon Framework ne permet pas de passer plusieurs repositories, donc je ne peux pas séparer les différentes colonnes.
	Exemple:
	`HttpServer::new(move || {`
	`App::new()`
	`//Définition des données`
	`.app_data(db_data.clone())`
	`.service(index)`
	`.service(create_student)`
	`…`
		J'ai simplement mis toutes les opérations de MongoDB dans le même repository pour pouvoir en déclarer un seul. J'ai ensuite séparé le repo en sections pour que ce soit plus clair. On pourrait aussi créer des fichiers séparés pour les méthodes mongo qui seraient appelées par le repository central.

Visual Studio Code ne lit pas bien le code Rust	Visual Studio Code marque des erreurs dans mon code Rust, comme s'il ne le reconnaissait pas. Notamment sur la déclaration des routes comme suit:
	
	Exemple:
	`//Cette partie donnait toujours des`
	`//erreurs`
	`#[post("/student")]`
	`pub async fn create_student …`
		J'ai du downloader plusieurs extensions de Visual Studio Code pour être capable de bien détecter les erreurs. Il ne fait malheureusement pas d'autocomplétion et n'arrive pas à détecter les variables, mais détecte au moins le bon format.

Format des chiffres non fonctionnel	Je n'arrivais pas à mettre un chiffre dans mes données. J'utilisais le format Decimal128 comme ceci:
	`pub struct Note {`
	`#[serde(rename = "_id", skip_serializing_if = "Option::is_none")]`
	`pub id: Option<ObjectId>,`
	`pub student_id: String,`
	`pub evaluation_id: String,`
	`pub note: Decimal128,`
	`}`
		Decimal128 demande un format spécial qui est ensuite convertit en nombre. Ce n'était donc pas le bon format à utiliser dans ce cas-ci.
		Je n'avais pas trouvé car les types en Rust sont moins évidents que les autres langages. Il fallait utiliser le type i32 pour Integer 32 bits.

CONCLUSION

Le Rust, est, comme sa réputation l'indique, extrêmement rapide. Sur mon ordinateur, une requête à un API pour afficher du simple texte prend 3 millisecondes. C'est beaucoup plus rapide que tout ce que je connais.
Par contre, ce langage est très dur à prendre en charge. Il n'y a presque aucune méthode préfaite pour sauver du temps malgré l'utilisation d'un framework. La structure du code n'est pas non plus intuitive, c'est un langage qui mélange un peu les principes du TypeScript avec l'optimisation du C. Bref, dur à coder.
L'API répond à tous les buts visés, soit de pouvoir créer, modifier et supprimer des étudiants et des évaluations. Il permet aussi de donner une note aux étudiants à chaque évaluation. Mais la manière dont il a été implémenté est beaucoup moins claire que d'autres langages ne l'aurait fait.
En conclusion, je dirais que pour un système très simple qui a besoin d'une rapidité hors du commun, le Rust serait parfait. Mais je le déconseille pour tout projet moindrement volumineux dû à sa difficulté.

RÉFÉRENCES

Repo Github du framework d'Actix Web: https://github.com/actix/actix-web

Site officiel d'Actix Web: https://actix.rs/

Tutoriel pour faire un API de base avec Actix Web:
https://dev.to/hackmamba/build-a-rest-api-with-rust-and-mongodb-actix-web-version-ei1

Github du framework Tower-Web: https://github.com/carllerche/tower-web/issues

Test de vitesse entre différents frameworks d'API en Rust:
https://github.com/rousan/rust-web-frameworks-benchmark

Github du framework warp: https://github.com/warpdotdev/Warp

Github du framework axum: https://github.com/tokio-rs/axum

Github du framework Rocket: https://github.com/SergioBenitez/Rocket

Site officiel de Rust: https://www.rust-lang.org/