Technical Report 1

Authors

Uyen Tran (Sara)

Aseem Thakar

Phillip Ebdon

Andy Zhan

Weilun Jason, Toh

FIT3170 Semester 2-Summer Semester, 2020

Monash University Clayton

Abstract

This technical report aims to analyse different technical alternatives for building a web-based tutor allocation system as requested by the client and to give recommendations based on the client and the system’s requirements as well as the team technical experience. Analysis was done for the backend framework and language, database technology, object-relational mapping (ORM) library and backend testing framework.

Node.js is recommended as our backend framework because of the system's low computational need, the framework popularity and simplicity as well as its use of JavaScript/TypeScript, making it easier for developers in our team to take on fullstack development. TypeScript is our recommended language despite being a steeper learning curve compared to vanilla JavaScript, its benefits in providing static types, compile time errors and autocompletion can help enforcing better code quality and readability and enable faster development.

For our database, since the system data volume is not extremely huge and that data has clear structures and relationships, SQL is a more suitable choice compared to NoSQL. Any of the SQL popular implementations on the market right now should be sufficient for our needs so PostgreSQL is chosen for its true free and open-source nature while providing some flexibility in terms of functionality.

TypeORM is the recommended ORM library for Node.js because of its good integration with TypeScript, popularity, readability and ease of use.

For the testing framework, our recommendation is Jest for its popularity, support, relatively simple setup and configuration with useful inbuilt tools.

Table of Contents

Domains

Backend

Framework

The back end framework is responsible for handling the business logic of a web application. It is particularly important the chosen framework is fast and scalable while being easy-to-use and a popular choice among developers.

In our use case, speed and scalability are important due to the sheer size of the Monash University organization; this application will likely be used by hundreds of staff and thus needs to be able to meet high volumes of requests while maintaining high throughput. The tutor allocation use case also demands a certain ease-of-use and popularity to make it frictionless to hire developers to work on and maintain the project when FIT3170 concludes.

Alternatives

In this endeavour, there are three main alternatives listed below that fit the above criteria. All three solutions can be considered fast/scalable, easy-to-use and relatively popular in the developer community. However, they do have notable differences which are explored below which impact one (or more) of the aforementioned criteria.

Node.js

Node.js applications are written in JavaScript [1], allowing for easy transitions between front and back end development with little context switching required. Additionally, Node.js' event loop/callback architecture [2] makes development easy and scalable with less boilerplate code needed and efficient request handling [3].

However Node.js does have limitations. For one, it's not truly multi-threaded [2], meaning use cases that require a lot of business logic/computations (instead of basic CRUD applications) may result in request bottlenecks. Node.js also lacks a clear, opinionated structure which could result in productivity losses (or alternatively provide flexibility) [4].

Django

Django applications are written in Python [5], which is another popular language (even exceeding JavaScript's popularity [6]). Django provides a lot of help and guidance for projects; it comes standard with libraries to handle ORM, routing, security and administration [7] (unlike Node.js which requires these to be added in separately). Also unlike Node.js, Django, with Python, is capable of using multiple threads [8], meaning use cases requiring heavy computational work won't result in bottlenecks.

However, Django's advantages have complementary disadvantages. Its structured approach to development results in a lack of flexibility around project structure and a lot of boilerplate

code. Its multi-threaded architecture may also be let down by Python's slow performance compared to JavaScript (in Chrome's V8 Engine) [9].

Ruby on Rails

Ruby on Rails applications are written in Ruby which is not as popular/declining in popularity compared to Python or JavaScript [6]. Its benefits are quite similar to those of Django but Ruby on Rails doubles down on its adherence to project structure and standards [10]. If developers get used to them, code can be more readable and more efficient in the long run.

However, these advantages are potentially outweighed by Ruby on Rails' lack of flexibility and its performance. The heavy standards and poor boot and runtime performance [11] introduce a high barrier of entry to a Ruby on Rails project compared to Node.js and Django. And its popularity is declining compared to the aforementioned frameworks as well, meaning developers may be in shorter supply in future.

Limitations of analysis

One topic of particular importance that wasn't accounted for in this analysis was Monash's prevailing tech stack. If the back end frameworks that Monash used across most of its internal systems were known, the decision may have been simpler so as to keep Monash's systems/hiring requirements consistent throughout the organization.

Language

When selecting a framework we are also selecting a language or set of to develop in. The choice of language has a range of effects such as performance, typing, packages/libraries available, and difficulty of entry based on the groups experiences.

Taking into consideration the team's experience, Ruby is unlikely to be chosen as we would be attempting to construct a product while also learning the language from scratch, on top of Ruby on Rails.

The group is experienced with Python, but using Python does come with some performance issues. Python does support multi-threading however Python does often suffer from performance issues and multi-threading can't always solve those issues. It would also be likely we'd need to introduce JavaScript for interactivity as Python doesn't include much support for client-side scripting, while there are plugins and such to allow browser side python, it does mean more dependencies.

JavaScript has been the browser side language for years and is the default for most web-dev. The group has some experience with JavaScript while not as much as Python; some members of the group do however have extensive experience and that allows members to educate each other on the language.

Javascript has a unique debate on whether to use Typescript or not. Typescript is one of the most used javascript language extensions as it forces the developers to strictly type the variables, attributes, objects, etc. Javascripts untyped nature can lead to a whole host of problems but also allows for a host of flexibility.[12] Typescript also requires compiling into JavaScript which is a positive as it can make debugging faster at build-time, but does add

more steps in development. Typescript also requires packages for type definitions, hence isn't ideal for lightweight applications. Typescript is continuing to gain a lot of support but many frameworks and tools still don't fully support TypeScript, so if chosen even more consideration needs to be spent on Framework and Package selections.[13]

It is agreed on by most sources [12], [13], [14] that TypeScript is a better option as any project becomes larger but it is advised not to let a false sense of security come from using TypeScript alone.

Notes on Backend and Language based on Spikes

To help make a decision not just based on perks of the language and frameworks, but also on the potential learning curve, and comfort of the team, a joint sample project was constructed showing off the initial preferences for technologies.

These were Node.js with Express for backend, and React for frontend. A typeORM database using SQLite was later added as proof of concept. This was all done in TypeScript (TS).

The response from this spike was varied, with those with more prior experience quickly picking up new tools, while the curve for those with less experience may be steeper. However the technologies chosen have lots of documentation and tutorials available, and with a variety of experience to assist with building a base to work off. With that in mind the selection of React, Node.js with Express, and TS, had general consensus.

Recommendation

Given the low computational complexity of the Monash tutor allocation system, Node.js seems like the best candidate. Its use of JavaScript means full stack development and integration is much easier than alternatives like Python and Ruby. And despite lacking multi-threading support, its event-loop architecture will be more efficient and scalable than that of Django and Ruby on Rails' for this use case. Finally, because Node.js is relatively popular, finding Node.js developers will be an easy endeavour even after FIT3170's completion.

The project is likely to be taken on in the future so JavaScript or Python would be suited for hiring, however selecting JavaScript with TypeScript grants the development benefits from using a strongly typed, compiled, and Interface orientation can help find issues faster, and increase readability to a larger group. Should also be noted the TypeScript community is small but increasing, hence improving the teams skills in a growing area is also beneficial.

Database

We have decided that persistent storage is needed for our system since it has been proven to be difficult and performance limited to store and edit information on spreadsheets.

A few factors that we will need to take into account when choosing the database technology for our system are:

- The scale and performance of our system. We are expected to service approximately 1000 users during peak periods and from the sample given by the client, there are approximately 12000 allocations in a year.
- The team experience with the technology.
- The client's preference that the technology is popular enough for easy future hiring.

SQL vs. NoSQL

The main differences between SQL and NoSQL are [15]:

	SQL	NoSQL
Pros	● Reduced data storage footprint ● Data integrity (ACID) ● Standard language: SQL ● Complex and more flexible queries ● Team experience	● Vertically scalable ● Highly available, no significant point of failure ● High performance for high volume data ● Suitable for unstructured data
Cons	● Rigid data models and schemas ● Difficult to scale horizontally ● Single point of failure	● Vague interpretations of ACID constraints ● Distributed system problems

The advantages offered by NoSQL databases are not very applicable to our system requirements. For instance:

- Our system is not expected to be distributed since the most of the users are in Melbourne and the client requires that the data to be stored in Australia.
- Our data structure is not expected to change drastically in the future. The data has clear structure, entities and relationships between them so they do not benefit from the flexibility of NoSQL
- Our data volume is not 'big data' and thus the performance benefit from NoSQL is not very substantial
- The system does not require high availability since it is mostly used only at the beginning of teaching periods.

- The strict consistency and transaction offered by SQL databases is also important for our system since changes to allocations, such as swapping, should be correctly reflected at any time.

Overall, we think that SQL is a better fit for our requirements.

SQL implementations

MySQL

Some of MySQL benefits and characteristics are [16]:

- Free and open-source, while being owned and supported by Oracle
- Matured and popular with a huge community, extensive testing and stability
- Compatible with major platforms
- Sharding (horizontal partitioning). This is particularly interesting since this feature is usually not supported by many other SQL databases. This feature can help alleviate the potential performance bottleneck of SQL databases.

Oracle database

Some of Oracle database benefits and characteristics are [16]:

- Professionally developed and managed
- Has a unique SQL “dialect”
- One of the more expensive options
- Compatible with major platforms

Microsoft SQL Server

Some of Oracle database benefits and characteristics are [16]:

- Professionally developed and managed
- Has a unique SQL “dialect”
- Less costly than Oracle database
- Compatible with major platforms

PostgreSQL

Some of PostgreSQL database benefits and characteristics are [16]:

- Free and open-source. The PostgreSQL Global Development Group develops and manages the system.

- Compatible with major platforms
- Object Oriented Database Management System (ORDBMS). PostgreSQL supports some object oriented features such as table inheritance and function overloading. This offers the benefits of both a strictly relational model (SQL) and a strictly object-oriented model (NoSQL).
- High ACID compliance. PostgreSQL is known for offering the highest levels of atomicity, consistency, isolation, and durability.
- No customer support but has active community support. There are paid third-party service providers if a higher level of support is needed.
- Uses pure SQL

Recommendation

Given that our system is at a relatively small scale and is in its early development stage and thus we are not likely to need the professional support and proprietary features offered by Oracle and Microsoft, MySQL and PostgreSQL seem to be the most suitable choice for our team. These two databases are free and open-source and offer similar functionalities and general performance.

PostgreSQL has a few advantages over MySQL such as [17]:

- Better concurrency
- Strict ACID compliance
- Supports for object-oriented features
- Default PostgreSQL installation generally works better
- More extensible, supports more advanced data types and user-defined types
- Truly open-source while MySQL may have some licensing issues

MySQL advantages over PostgreSQL are:

- More popular with more third-party tools
- Generally less memory intensive

With these, we think that PostgreSQL benefits marginally outweighs MySQL and thus it is our choice. It should also be noted that since we decided to use an ORM, especially one that supports both MySQL and PostgreSQL, the switch between these two databases should not be too difficult if we need to.

Object-relational mapping tool (ORM)

Object-relational mapping is a programming framework that allows developers to write database queries and actions using the object-oriented paradigm of the developers preferred programming language.

SQL QUERY [18]

```
SELECT * FROM users WHERE email = 'test@test.com';
```

ORM QUERY [18]

```
var orm = require('generic-orm-library');
var user = orm("users").where({ email: 'test@test.com' });
```

The above example perfectly illustrates the purpose of using an ORM. CRUD operations can be written in syntax that is similar to the chosen language of programming. With all it's prowess, ORMs do have certain drawbacks.

Here is a table comparing the pros and cons of using an ORM:

Pros	Cons
Great for teams with a mixed knowledge of SQL ● Unrealistic to expect all members to have SQL knowledge ● An ORM levels the playing field and brings the team up to speed quicker	Could be a hindrance to developers with deep expertise in SQL ● ORM translations introduces additional overhead ● A proficient SQL developer could write more efficient SQL code most of the time.
Database technology is abstracted ● Switching between systems such as PostgreSQL to MySQL is simple	Abstraction ● Since SQL is abstracted away by the ORM, developers may not understand what is going on under-the-hood which is detriment to individual learning of database systems
More advanced ORMs provide many advance features which may be useful for the project down the line	
	Additional overhead ● Initial configuration of an ORM introduces additional overhead to project

	• Additional overhead is also required to learn basics operations of any given ORM

ORM Libraries for Node.js

Sequelize

Some of Sequelize features and support are:

- Supports MySQL, Microsoft SQL, PostgreSQL (but not Oracle)
- Most popular Node.js ORM with 23k stars and 3.6k forks on GitHub
- Promise-based Node.js ORM
- Works well with vanilla JavaScript but very basic TypeScript support
- Steep learning curve; Not the best Documentation

TypeORM

Some of TypeORM features and support are:

- Supports MySQL, Oracle, Microsoft SQL, PostgreSQL
- Second most popular Node.js ORM with 21k stars and 3.6k forks on GitHub
- Promise-based Node.js ORM
- Much better integration with TypeScript
- Steep learning curve; Not the best Documentation
- QueryBuilder allows developers to build complex SQL queries using convenient syntax

Simple example of `QueryBuilder`:

```
const firstUser = await connection
    .getRepository(User)
    .createQueryBuilder("user")
    .where("user.id = :id", { id: 1 })
    .getOne();
```

It builds the following SQL query:

```
SELECT
    user.id as userId,
    user.firstName as userFirstName,
    user.lastName as userLastName
FROM users user
WHERE user.id = 1
```

and returns you an instance of `User`:

```
User {
    id: 1,
    firstName: "Timber",
    lastName: "Saw"
}
```

Prisma

Some of Prisma features and support are:

- Supports MySQL and PostgreSQL (but not Oracle and Microsoft SQL)
- Low adoption with 6k stars and 302 forks on GitHub
- Supports TypeScript
- Less boilerplate code required
- Technically still a beta product
    - Stability and long term support is questionable

Recommendation

As the team has chosen PostgreSQL as the database system and TypeScript as the language of choice, PostgreSQL and TypeScript support will be our main concern when selecting an ORM. This eliminates Sequelize as it has poor TypeScript support. With Prisma and TypeORM remaining, we have selected TypeORM as our ORM of choice as Prisma is a beta product and the team has deemed the risk of using a beta product in this project too high.

With TypeORM, the team has accepted several risks that comes with it:

- Initialisation overhead
- Learning curve stemming from poor documentation

These risks were deemed acceptable with the benefits that TypeORM brings.

Testing framework

We considered the following criteria for our backend testing framework: ease of use, ease of integration, level of support, and range of features.

Jest

One of the frameworks we considered was Jest, a JavaScript test runner. Jest is widely recommended as an easy-to-use testing framework: it uses simple syntax and contains less boilerplate code (compared to other popular frameworks like Mocha). It also contains built-in support for TypeScript [19].

It is currently listed as the most popular JavaScript testing framework [20], and continuous updates and improvements are regularly released; hence, it is very likely that it will remain strongly supported for a long time.

Jest also features built-in support for promises, matchers, and mock functions/manual mocks, as well as parallel test running for improved performance [21].

Mocha

Mocha is a testing library for Node.js, designed to be simple, extensible, and fast.

When it comes to ease of use/integration, Mocha may take longer to set up compared to Jest due to its reliance on external libraries for functionality such as assertion or mocking libraries. However, this is also what lends Mocha its high degree of flexibility and customisation. As for syntax, the syntax used in Mocha is almost identical to that of Jest (as they are both based on Jasmine syntax).

Mocha also boasts a large community following with lots of tools and utilities; it is currently the second-most depended-upon module on npm. [22]

Like Jest, it has support for things like asynchronous testing, but as previously mentioned, it lacks built-in support for features like code coverage, assertions and mocking. [23]

Ava

Ava is a relatively new, lightweight test runner for Node.js.

It is simple and fast to set up, although its minimalistic, barebones approach means that a more detailed knowledge of testing is required to get the most out of it.

Its recency also means that not a lot of documentation or tutorials exist for Ava, especially compared to long-standing frameworks like Mocha.

Ava, like Jests, runs tests in parallel, meaning that it is likely to be faster than Mocha, which runs tests serially without process isolation. However, it has no built-in code coverage either, and its simplistic approach means that separate libraries are needed for features such as mocking. [24]

Recommendation

Our recommendation for the testing framework is Jest. The main reason we chose to recommend Jest is due to the priority we placed on ease of use: given the amount of time we have to develop this, and our skill level as students, we figured that Jest would give us the ability to get started more quickly, as it appears to be widely recommended for how easy it is to use. Jest also has important features such as coverage reporting mocking built in, whereas with Mocha or Ava, external libraries are needed and we would have to spend precious time manually choosing libraries. Jest also works well on the frontend, especially with React applications, so the possibility of using the same framework to test both the frontend and the backend is a large bonus.

Conclusion

Based on our analysis, our recommendations for the system are as follows:

Area of concern	**Recommendation**
Backend framework and language	Node.js + TypeScript
Database	PostgreSQL
ORM	TypeORM
Backend testing	Jest

The recommended choices are popular per the client's preference and we have also taken into account other factors such as the performance and computational load required, the expected scale of the system and the team overall experience with these technologies. We expect these choices to be sufficient in fulfilling the system business requirements and integrate well with each other.

Bibliography

[1] "The V8 JavaScript Engine", *The V8 JavaScript Engine*, 2020. [Online]. Available: https://nodejs.dev/learn/the-v8-javascript-engine. [Accessed: 24-Oct-2020]

[2] "The Node.js Event Loop", *The Node.js Event Loop*, 2020. [Online]. Available: https://nodejs.dev/learn/the-nodejs-event-loop. [Accessed: 24-Oct-2020]

[3] "How The Event Loop Works In Node.js", *Heynode.com*, 2020. [Online]. Available: https://heynode.com/tutorial/how-event-loop-works-nodejs. [Accessed: 24-Oct-2020]

[4] H. Di Francesco, "Advantages and Disadvantages of Node.js", *Code with Hugo*, 2020. [Online]. Available: https://codewithhugo.com/node-pros-and-cons/. [Accessed: 25-Oct-2020]

[5] "The Web framework for perfectionists with deadlines | Django", *Djangoproject.com*, 2020. [Online]. Available: https://www.djangoproject.com/. [Accessed: 25-Oct-2020]

[6] "Stack Overflow Developer Survey 2019", *Stack Overflow*, 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019#key-results. [Accessed: 25-Oct-2020]

[7] "Django at a glance", *Django Docs*, 2020. [Online]. Available: https://docs.djangoproject.com/en/3.1/intro/overview/. [Accessed: 25-Oct-2020]

[8] J. Anderson, "An Intro to Threading in Python – Real Python", *Realpython.com*, 2020. [Online]. Available: https://realpython.com/intro-to-python-threading/. [Accessed: 25-Oct-2020]

[9] K. Martin, "Python vs JavaScript: Which One Can Benefit You The Most?", *Habr.com*, 2019. [Online]. Available: https://habr.com/en/post/471134/. [Accessed: 26- Oct- 2020]

[10] D. Hansson, "Ruby on Rails Doctrine", Ruby on Rails, 2016. [Online]. Available: https://rubyonrails.org/doctrine/. [Accessed: 26- Oct- 2020]

[11] "A Comparative Study of Two Frameworks - Ruby on Rails vs Django", Ongraph, 2020. [Online]. Available: https://www.ongraph.com/ruby-rails-vs-django-better-framework-web-application/. [Accessed: 26- Oct- 2020]

[12] “https://hackr.io/blog/typescript-vs-javascript” , Hackr.io, 07 Aug, 2020 [Online],

[13] “TypeScript vs. JavaScript: Should You Migrate Your Project to TypeScript?”, Stackify, Sept 2017, [Online]. Available: https://stackify.com/typescript-vs-javascript-migrate/ [Accessed: 27-Oct-2020]

[14] “Typescript vs JavaScript: What's the Difference?”, Guru99, [Online].

https://www.guru99.com/typescript-vs-javascript.html [Accessed: 27-Oct-2020]

[15] "SQL vs. NoSQL Databases: What's the Difference?", *Ibm.com*, 2020. [Online]. Available: https://www.ibm.com/cloud/blog/sql-vs-nosql. [Accessed: 06- Nov- 2020]“

[16] M. Smallcombe, "SQL vs NoSQL: 5 Critical Differences", *Xplenty*, 2020. [Online]. Available: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/. [Accessed: 06- Nov- 2020]

[17] "MySQL vs PostgreSQL -- Choose the Right Database for Your Project", *Okta Developer*, 2020. [Online]. Available: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres. [Accessed: 06- Nov- 2020]

[18] “What is an ORM and Why You Should Use it”, Bitsrc.io, [Online]. https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a [Accessed: 5 Nov 2020]

[19] S. Bekkhus, "Jest 24: Refreshing, Polished, TypeScript-friendly · Jest", Jestjs.io, 2020. [Online]. Available: https://jestjs.io/blog/2019/01/25/jest-24-refreshing-polished-typescript-friendly. [Accessed: 01- Nov- 2020]

[20] "The State of JavaScript 2019: Testing", *2019.stateofjs.com*, 2019. [Online]. Available: https://2019.stateofjs.com/testing/. [Accessed: 02- Nov- 2020].

[21] "Testing Asynchronous Code · Jest", *Jestjs.io*, 2020. [Online]. Available: https://jestjs.io/docs/en/asynchronous. [Accessed: 02- Nov- 2020].

[22] "Libraries - The Open Source Discovery Service", *Libraries.io*, 2020. [Online]. Available: https://libraries.io/search?order=desc&platforms=NPM&sort=dependents_count. [Accessed: 05- Nov- 2020].

[23] "Jest vs Mocha: Which Should You Choose?", *Medium*, 2020. [Online]. Available: https://blog.usejournal.com/jest-vs-mocha-whats-the-difference-235df75ffdf3. [Accessed: 05- Nov- 2020].

[24]"JavaScript unit testing frameworks in 2020: A comparison | Raygun Blog", *Raygun Blog*, 2020. [Online]. Available: https://raygun.com/blog/javascript-unit-testing-frameworks/. [Accessed: 05- Nov- 2020].