CSE P 544 Project Report

How does Using Backend-as-a-Service Affect Web App Design?

Cosmo Wang

1 Problem Description

A trend in web application development in recent years is that web developers are favoring Backend-as-a-Service (BaaS) when they're doing light-weighted web development. These services provide a way for web developers to store and access data from their front-end application without needing to implement a fully functional back-end server, including a database. Despite that obvious benefit of using such a service, this mini-project investigated how using a BaaS provider affects web applications from a system design perspective, when compared to using a traditional back-end server with a database.

2 Systems Used

There exist many popular BaaS providers. This project experimented one of the providers, Back4App. Back4App is a BaaS provider that is based on NodeJS, Parse Server, and MongoDB. It provides a easy-to-use interface for developers to upload and store data in tables, and then makes the data accessible through Parse Server API from developers applications. In this project, a fully functional web application is implemented using Back4App as its backend. On the other hand, the same web application is also implemented using Node.js server and PostgreSQL database.

3 Application Overview

The application used in this project is a Japanese anime data website. For a user of the application, animes are categorized into two categories: new and old animes.

New animes are animes that the user watches on TV weekly in the previous season, the current season, and the next season.

Old animes are animes that the user is watching, has watched, or want to watch after they have been fully released. The user can store a list of old animes with user's personal ratings. The user can also store a list of new animes that the user watches as they are released each week. Once the user finishes watching a new anime, the user can convert the new anime into an old anime. New animes are supposed to be removed two seasons after it was broadcasted on TV. Additionally, the user can also store a list of quotes from various animes that the user likes.

In order to serve the functionality described above, the following data needs to be stored on the server: (See Table 1, 2, 3 in Appendix for a full list and descriptions of the attributes.)

- Common attributes for both new and old animes

- New anime specific attributes
- Old anime specific attributes
- Content and metadata of quotes

Given exactly the same data to store and serve, this project implemented two servers that support the same front-end interface.

4 Using Back4App as Backend

4.1 Server Structure Overview

Using Back4App, developers stores their data in classes. Each class can have any number of columns, where each column can store data of a specific type, which is very similar to a traditional SQL table. However, classes don't support the notion of key, foreign key, and value uniqueness in a column.

Using this class-based struture, the data described in the previous section was stored in three classes: OldAnimes, NewAnimes, and Quotes. OldAnimes class contains all common attributes and old anime specific attributes. NewAnime class contains some common attributes and all new attribute attributes. Quotes class contains all content and metadata of quotes.

After storing data inside the three classes, Back4App automatically provides Parse Server APIs for developers to access and update those data from front-end.

4.2 Design Analysis

The hardest design decision made for the above schema was the separation between new and old animes.

As described in the previous section, new and old animes share many attributes and a particular anime can be both a new and an old anime at the same time. Therefore, if we want to avoid data duplication, then it's tempting to store both old and new animes in a single class. However, since new and old animes each has specific attributes, combining them into one table would result in many null values in many columns. Moreover, Back4App classes don't have a unified null value for different data types and dealing with different null-equivalents for different data types would be a big pain. So the decision was to partially separate new and old animes into two classes.

Given this design decision in terms of data storage, in the front-end UI, new animes and old animes data can be fetched separately from server and are displayed in separate views. But a drawback of this design is that the new animes view on the front-end can only display attributes from the NewAnimes class. If we want to display all information about a new anime, we need to double the loading time to make another server call to access the old anime table. In addition to the additional server access, to look up each new anime in the old anime class, we have to do text comparison on anime names, because, again, Back4App don't support the notion of key and foreign key, so it's hard to relate one class with another.

Finally, since data is fetched directly from Back4App to front-end, all data formatting and processing need to be done on the front-end, which unnecessarily complicated the front-end code.

4.3 Conclusion

To summarize the experience with Back4App, although having a easy-to-use interface to manage data and not having to implement a server sound great, but the class-based database that Back4App provides has some clear drawbacks, such as can't easily join one class with another and no unified null value for all data types. As a result, the front-end design is dominated by the limitation of the is limited and complicated by these drawbacks.

5 Using Node.js and PostgreSQL as Backend

5.1 Server Structure Overview

After designing the database schema for Back4App, the most natural design choice for the PostgreSQL is to relate new and old animes using foreign keys. Therefore, the database was built under the following ER diagram:

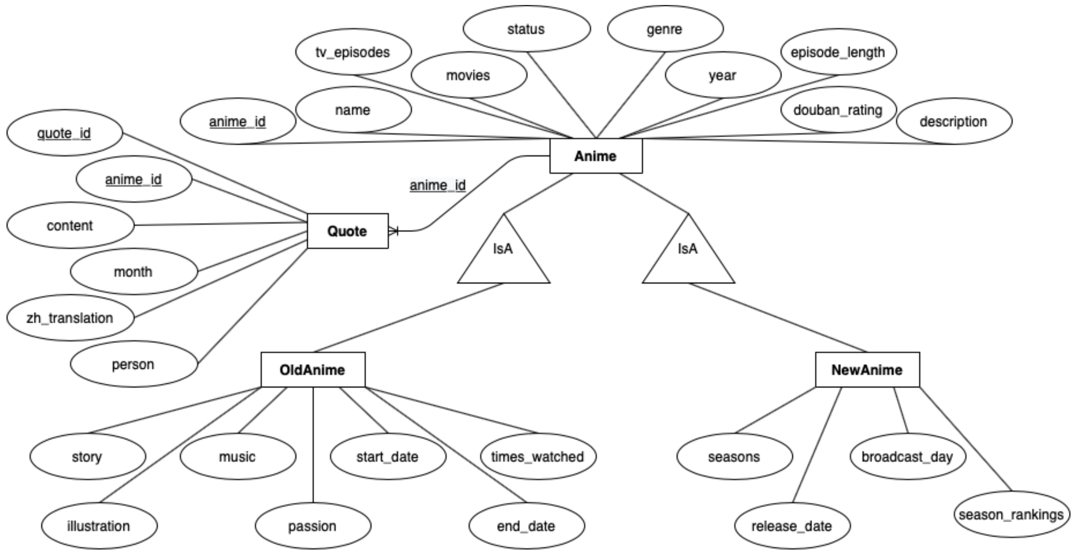

Figure 1: ER Diagram

After importing all data into the database, a server was written using Node.js to support whatever kind of request needed by the front-end application. Details about the server is omitted in this report since they are mostly irrelevant.

5.2 Design Analysis

By using anime_id as a foreign key in NewAnime and OldAnime tables, data stored in the database is never duplicated and also don't contain unavoidable null values. Moreover, front-end can fetch complete information of both new and old anime using one server call by joining NewAnime and OldAnime tables with the Anime table. An additional benefit is that the Quote table can also be joined with the Anime table to display more information on the front-end optionally. These benefits mainly comes from the fact that traditional SQL databases are built on top the a relational model, which makes it easy to relate tables with each other.

Additionally, the existence of an intermediate server between front-end and the database takes away a lot of data formatting task from the front-end, which greatly increases the maintainability of the front-end code. Moreover, tasks can be scheduled to run periodically on the server. Therefore the removal of new animes two seasons after they were broadcasted on TV can be done automatically.

5.3 Conclusion

In summary, using a traditional SQL database and a server as back-end for an web application releases the full potential of the application by giving fine-grained control of the data storage and access pattern to developers. Although developers need to implement more in this case, the ability to request data in any desired format from front-end puts no restriction on what can be displayed on the front-end to users.

6 Conclusion

BaaS providers makes it easier for developers to deploy a web application by combining database and server and exposing the data to front-end. However, these providers typically restrict how data are stored and will eventually complicate and dominate front-end design. Moreover, front-end code will gradually become dependent on the particular BaaS provider, which makes it hard to port the same front-end code to use a different provider when necessary. On the other hand, building web applications using traditional SQL database and a separate server puts no restriction on front-end design. Therefore user experience becomes the dominate factor of the application design process. Additionally, since the server acts as an adapter between front-end and database, front-end code can stay the same when database needs to be changed, which is a generally a better practice in terms of system design.

7 Appendix

Database and server code is here. Front-end code and demonstration is purposely omitted in the report due to personal data privacy.

Common attributes for both new and old animes:

Table 1: Common Anime Attributes

Attribute Name	Attribute Definition
name	name of the anime
year	year when the anime was released
genre	genre of the anime
tv_episodes	number of episodes of the anime, including OVAs and OADs
episode_length	length in minute of one episode
movies	number of movie versions of this anime
status	want to watch, watching, or watched
douban_rating	rating of the anime on a Chinese rating website, Douban
description	description of the anime

New anime specific attributes:

Table 2: New Anime Attributes

Attribute Name	Attribute Definition
seasons	season(s) when the anime is broadcasted on TV in Japan
release_date	first day when the anime is broadcasted
broadcast_day	day of the week when the anime is planned to be broadcasted
season_rankings	user's ranking of the anime in a season among all other animes broadcasted in that season

Old anime specific attributes:

Table 3: Old Anime Attributes

Attribute Name	Attribute Definition
story_rating	user rating of the story of the anime
illustration_rating	user rating of the illustration quality of the anime
music_rating	user rating of the music quality of the anime
passion_rating	user's passion rating of the anime
start_date	date when the user started watching the anime
end_date	date when the user finished watching the anime
times_watched	number of times the user watched the anime