Bricked Up

Francesco Schenone Sebestyen Deak Kacper Grzyb Ignad Bozhinov
Leonardo Gianola

17-12-2024

1 Introduction

The LEGO investment market has emerged as a profitable opportunity, with an average return of 11% [1]. However, the current process for analyzing prices, trends, and data on LEGO sets needs to be more cohesive and efficient. To address this, *Bricked Up* centralizes key information, providing interactive graphs, historical price comparisons, and tools to explore investment opportunities efficiently. This report outlines the development of the software, inspired by the Bloomberg Terminal, and its role in simplifying LEGO set investments for users.

2 Front-End

This project's first task is to build your application's front-end side. This section should clearly describe the technical implementation of the work put into building the front-end:

- Technically describe the use of HTML 5: which HTML tags do you use, where, and why
- Technically describe the use of CSS: why and how you use CSS (including interesting selectors/declarations and how it is incorporated in the application)
- Technically describe the use of JavaScript: why and how you use it in your application (including interesting behaviors and how they are incorporated into the application)

Resources. Lectures 1 to 3.

Length. 2 columns.

3 Resource Management

Since we chose Laravel as our MVC framework, before we could define a resource management system in our project, we first needed a database. Our database solution of choice ended up being Supabase with PostgreSQL. We settled on it because it is very easy to set up, provides a generous free plan, has an intuitive UI, and can be easily scaled up by upgrading the database plan.
With the database set up, we created a schema for our application using draw.io (the ERD diagram can be found in the Appendix) and started writing Laravel migrations to implement the database. The central two models of our database are User and Set (stored in *users* and *sets* tables respectively) , the latter storing all of the most important data about a LEGO set. The rest of the models within our database center around adding additional typechecking or information to the sets, such as the set's price records. As of the current state of the project, we populate the database from multiple different sources, and using multiple tools, but this could be streamlined if this application were to evolve.

- **Seeders** - We defined some basic database seeders for including static data such as the testing admin account (to be removed for production) or set availability types
- **Python Web Scraper** - A simple python console program utilizing *Selenium* to scrape Brickeconomy for set themes and subthemes
- **Playwright Web Scraper** - The main tool for obtaining the current prices of the sets within

our database, an implementation of *Playwright* that scrapes eBay for price records

- **Admin Upload Data Page -** Albeit a temporary solution, this is currently the main tool for adding new sets into the database

Adhering to the MVC framework, we hydrate our views with data by the use of Laravel Controllers. Almost every single page has its own controller, so that the data can be custom formatted and optimized to the needs of that specific page. Our controllers perform different CRUD operations, and some of them can only be accessed by the admin user, as to comply with the project's requirements of user roles and application functionality. The operations include:

- **Account CRUD -** Before a user is logged in (checked by Laravel Breeze), they are only able to see our landing page with the ability to create an account. Most of the account management functionality was already pre-provided for us by Laravel Breeze, which made the development process a lot smoother, as we had to either user the pre-existing controllers or recycle their functions. Most note-worthy, the *RegisteredUserController* manages user account creation, and the *PasswordController* manages user authentication. The rest of the controllers within the Auth directory are used within the Settings page for editing the account details and deleting the account itself.

- **Set Details CRUD -** The admin-only Upload Data page allows the admin to upload CSV files that contain information about the set they want to add into the database. We created a small, initial dataset for our application, since we knew adding sets would require significant moderation. The data sanitization and creation is handled by the *FileUploadController*.

- **User Favourites CRUD -** In the settings page, a user can select their favourite sets, themes and subthemes from all the available ones in the database. The controller responsible for this functionality is *SettingsController*.

- **User Inventory CRUD -** From the settings page, a user is able to add a set to their own set Inventory, where they can see a summary of all the sets they own. This is handled by the *InventoryController*.

- **User Dashboard Layout CRUD -** The *DashboardController* is responsible for both providing the data for the dashboard view, as well as saving the user custom created dashboard within the Edit Dashboard Layout page.

4 Authentication and Authorization

The third task for this project was to create authentication and authorization capabilities. The idea was to have a landing page and information about the platform which was freely accessible but required user registration for the core functionality of the website. This section outlines the approaches used for user authentication, role-based access control, and session management.

Authentication.

We did not steer clear of Laravel's handy features and decided to implement the pre-built Breeze authentication, as already mentioned earlier, which was later extended for added security. The system supports two primary types of users: Regular Users and Admin Users. Regular Users can access basic features such as viewing LEGO investment data and trends, while Admin Users have additional privileges, such as managing investment records, overseeing user accounts, adding favorite sets, and more. Unregistered users are restricted from accessing any features and are blocked from the page's URL until they complete the registration process.

To accommodate our needs, the user model and migration were extended from Laravel's default ones, with several key modifications. A role field is added to distinguish between Regular and Admin Users. Additionally, the ”remember token” field is used to support the ”remember me” functionality, which allows users to remain logged in across sessions.

Another notable addition to improve security in the authentication process is two-factor authentication (2FA), which improves security by requiring users to enter a verification code sent via email during the login and registration processes. This step ensures that only authorized users can access the platform, adding an extra layer of protection against unauthorized access.

Upon successful login, the user's session is started, and the system applies middleware to enforce two-factor authentication, ensuring that the user has completed this step before granting full access. However, if the user fails to enter the correct code, a new unique one is sent to avoid code theft.

Authorization.

The platform's authorization system is based on role-based access control (RBAC), implemented through Laravel's middleware. The user's access level is determined through this role field with the help of middleware, playing a crucial role in enforcing access control. For example, only administrators are allowed to access routes that involve managing user accounts or approving investment listings. Regular Users can access general data and make investments but cannot modify any records or perform administrative functions.

Additionally, we use middleware to implement the 2FA, sessions, and inactivity logout. If a user is inactive for a specified period, they are logged out and prompted to log in again. The same is true if the user closes the browser tab or the session expires, with a message given 1 minute before being logged out. The user's activity is constantly tracked via mouse movement or changing pages.

Role Table.

To define the actions available to different user types, the table below shows the specific actions that Regular and Admin Users can perform:

Action	**Regular User**	
Register a new user	Yes	
Log in	Yes	
View LEGO investment data	Yes	
View and edit investment records	No	
Manage user accounts	No	
Approve/reject investment listings	No	
Access Admin Dashboard	No	
View and modify user roles	No	
Track session activity	Yes (via inactivity)	Y
Set favorite LEGO sets	Yes	
Access data export features (CSV/JSON)	No	
View system logs (user activity, errors)	No	

Table 1: Role-based Action Table

5 Conclusions

The goal of the conclusion is similar to the introduction: it summarizes the work itself and the takeaways a reader should take when reading this work. However, it can use the information presented in the work to be more specific than the introduction.

In the context of the Web Technologies course, the conclusion should clearly describe:

- Summary: summary of the work and main takeaways. Also include a class diagram of the system (the models).
- Future Work: interesting directions on how the presented work can evolve in the future (it may be the starting point to choose individual extension topics)

Length. Half a column.

References

[1] Dmitry B. Krylov, "LEGO investing as an alternative asset class: Annualized returns of 11%," *ScienceDirect*, 2021. Available at: ScienceDirect Article.