# TORUS: A digital asset marketplace

### Introduction

Discovering and sourcing 3D assets is an infamously difficult task for designers. Given the industry’s high-specialized nature, many freelance creators don’t have a platform where their digital products can easily be found.

The Torus Project aims to provide a platform which caters to 3D artists and designers from all backgrounds. With Torus, a character designer could easily find the work of a professional animator on the same platform, saving valuable time. This solution provides a marketplace where anyone with a particular niche – shader developers, texture artists, modellers and the like – can share or sell their creations.

Collaboration is the backbone of Torus’s model. After all, freelance creators and independent designers don’t have access to their own production team. The platform aims to enable anyone to licence content from other users with minimal friction.

### Analysis

The end users of this service are freelancers working on 3D asset design. Certain considerations were made in the design of Torus to accommodate those familiar with this industry, such as the inclusion of alternative asset categories such as shaders.

Client Research

Gathering data from those working in the field of 3D design was crucial for understanding the exact requirements of Torus’s intended market. As such, a brief interview was conducted with six freelancers working in this field. One common complaint expressed toward existing art asset distributors was a poor browsing experience. After asking about the experience of searching for required assets using similar services such as *CGTrader*, there were multiple accounts of high “visual noise” between listed items introducing confusion, which worsened the browsing experience. As such, consistent theming was a priority for a professional frontend. Another common complaint was a lack of ubiquity between the assets provided by these websites. The current solution of separate webservices for animations, meshes and shaders needlessly complicated the workflow of these freelancers, an opportunity for Torus to offer a more holistic 3D asset marketplace.

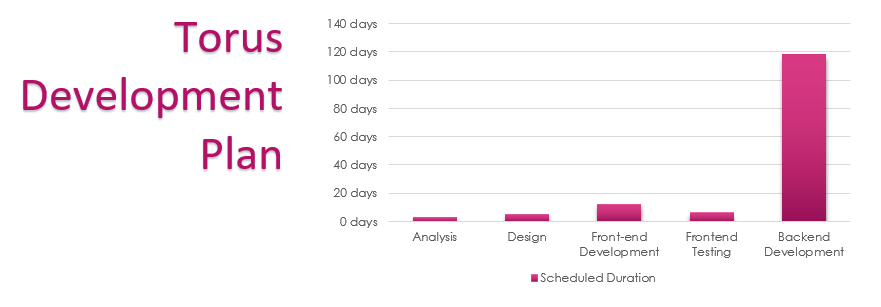
Relevant applications

As with any online marketplace, careful consideration was needed throughout the design process. To mitigate potential issues with copyright and third-party licensing, all product-related metadata and images were generated in-house for this project. However, some static content such as material font icons and internal javascript libraries (i.e jQuery and Bootstrap) were required for this website’s design requirements. To host this content locally without embedding the assets via an external Content Delivery Network (CDN), this code was selectively licensed from permissive, copyleft sources such as FontAwesome and the JS Foundation.

Privacy was also a key consideration in the design of Torus. The decision to host all required libraries without the use of Cloudflare and Google’s Content Delivery Networks was partly made to reduce the risk of persistent tracking across multiple webpages. The privacy community has expressed concern over the reliance on centralized CDNs for embedding libraries, so Torus has aimed to reduce unnecessary external connections. The application also avoids accessing the HTML canvas DOMRect and WebGL for realtime 3D rendering, unlike existing 3D asset marketplaces. Most similar web services offer a 3D preview of its user-hosted content along with WebXR rendering capabilities, both features which are strong browser fingerprinting vectors. As these features are frequently abused for web tracking, the Torus frontend has elected to offer a barebones pre-rendered image solution where users can be confident about the handling of their data. This also makes the site available to browsers which block WebGL by default, such as LibreWolf and the Tor browser.

Planning

The Torus project made heavy use of project management software for scheduling its numerous stages of development. Microsoft Project was used for this purpose, as it is an industry-standard project management solution favoured by software developers. As displayed in this bar chart, most resources and development time was spent on the backend implementation, as database management is the most crucial component of an online marketplace such as Torus.

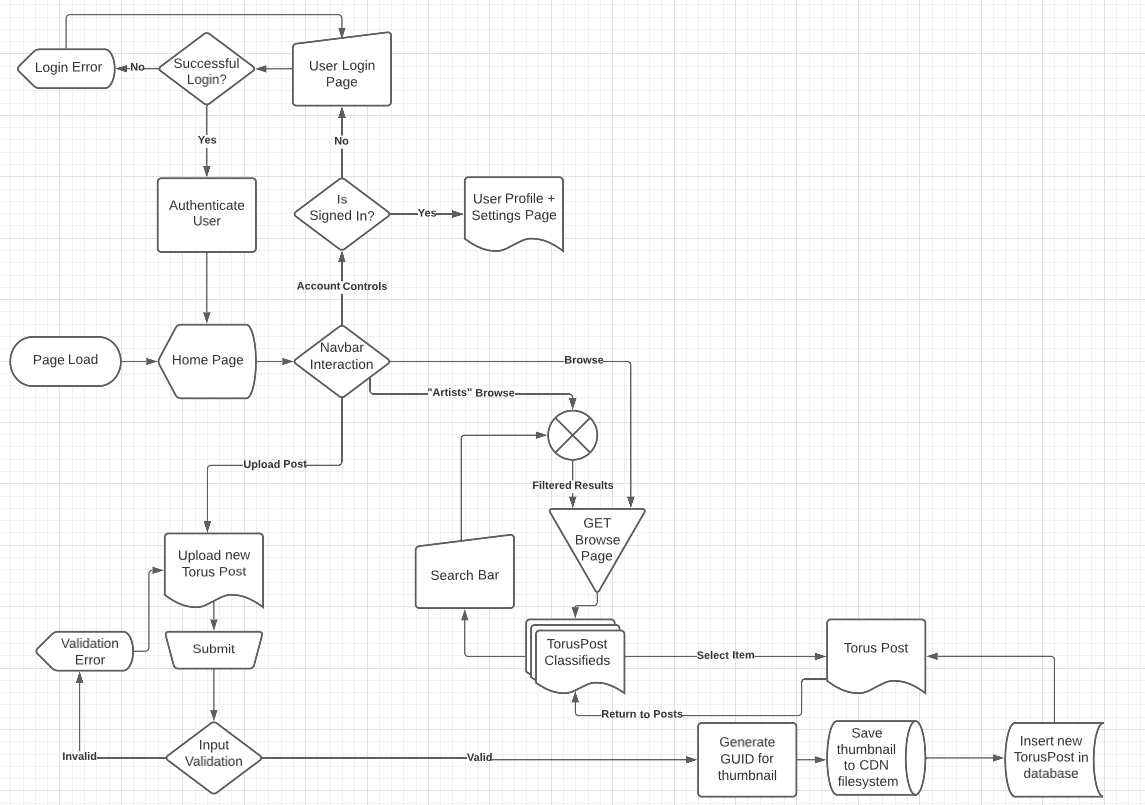


A Gantt Chart was also produced for the purposes of project time management. The first two weeks of the project were spent gathering feedback from 3D artists to directly contribute to the platform’s design. This required interviewing freelancers from numerous backgrounds; potential clients working in character design had more commercial expectations, whereas amateur mesh artists wished for a platform to share their work for free. These interviews provided time to integrate feedback into the project’s design schedule in its early stages of development.



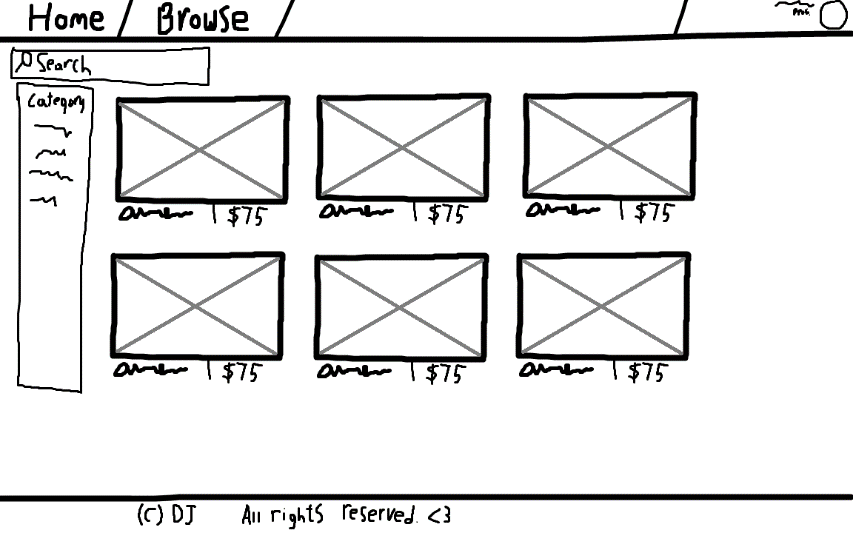
### Flow Chart

When designing a website’s workflow, it is important to plan how each user interaction flows into another. Inconsistencies and improper planning often lead to poor design decisions and a frustrating user experience. The Torus project uses a software flow chart to map out a user’s possible interactions, ensuring that no branches are awkwardly ‘dead-ended’. Extra care was put into the Torus platform to subtly guide the user back to the Browse page where suitable. As this page is where users will spend the most time, centring the site’s interactions toward the Browse page’s classifieds ensures that users do not feel lost when navigating the Torus platform.



### WireFrame

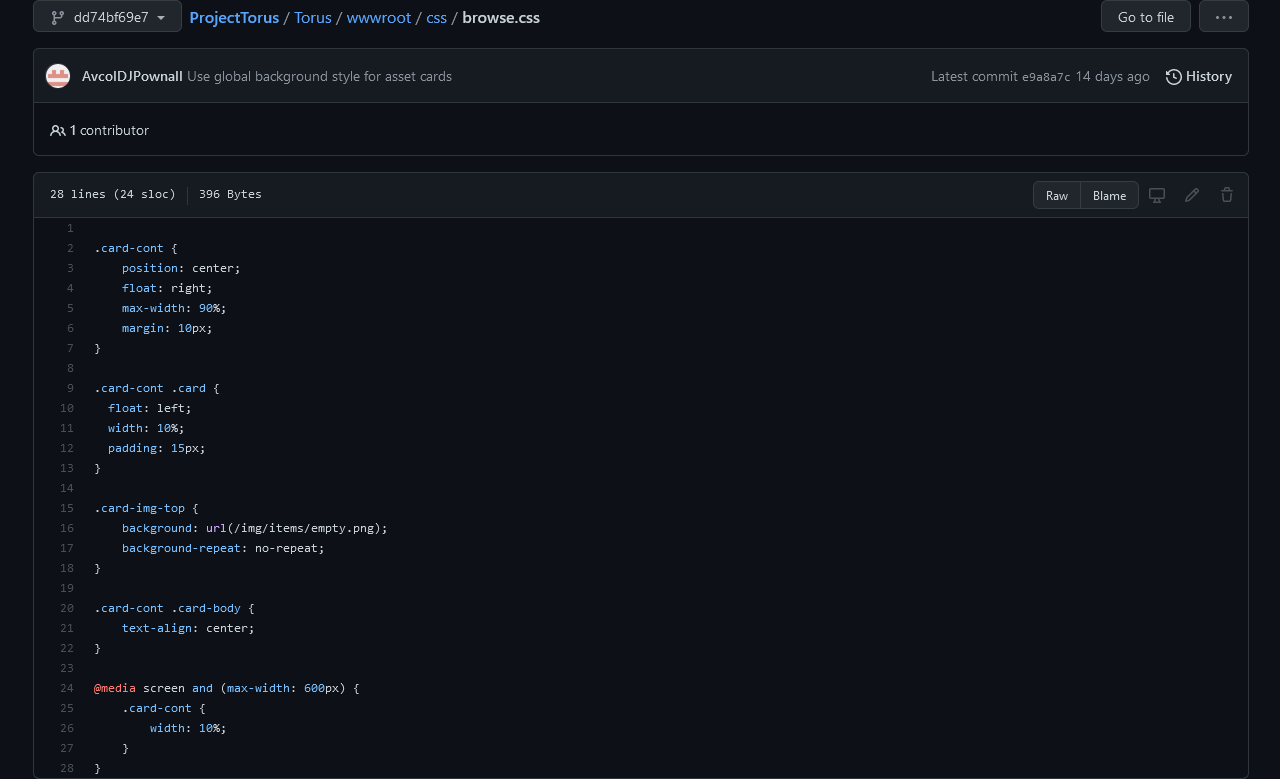
Consistent web planning and design are key to building an easy-to-use platform. The first iterations of Torus’s front-end design share many similarities with the finalized platform. Some aspects were modified to improve user experience, although placement of navigation items and the browse page’s layout have remained consistent through the platform’s development.



### Development & Testing

Throughout the development of Torus, various changes were made to the site’s front end and backend design to ensure the user experience would remain positive between devices.

Testing was conducted with a wide range of browsers and device display emulators. A large focus was placed around the Browse page, as this would be where the end user would spend the majority of their time. Edge’s Chromium-based browser showed good results at the browser’s default 1920x1080 resolution, but ran into inconsistent grid sizing if the user scaled their window below 75% or above 120%. To mitigate this, a fixed maximum box width was introduced in the page’s stylesheet:



This satisfied most situations with a 16:9 aspect ratio and enabled compatibility for Firefox’s resistFingerprinting.letterboxing setting. However, this change would later cause issues with some mobile devices. Certain screen aspect ratios were not able to render the grid’s elements without introducing gaps in the grid. After some consideration, the page’s styling was modified to maintain a dynamically scaled three column grid for mobile devices, while still using five columns in the desktop version. The number of displayed images was also raised to 60, a multiple of 2, 3 and 5. This would allow compatibility for devices of all aspect ratios, including edge-cases such as the Galaxy Fold.

As described by this test case diagram, file uploading and browsing required extensive testing to eliminate security issues while maintaining a positive user experience.

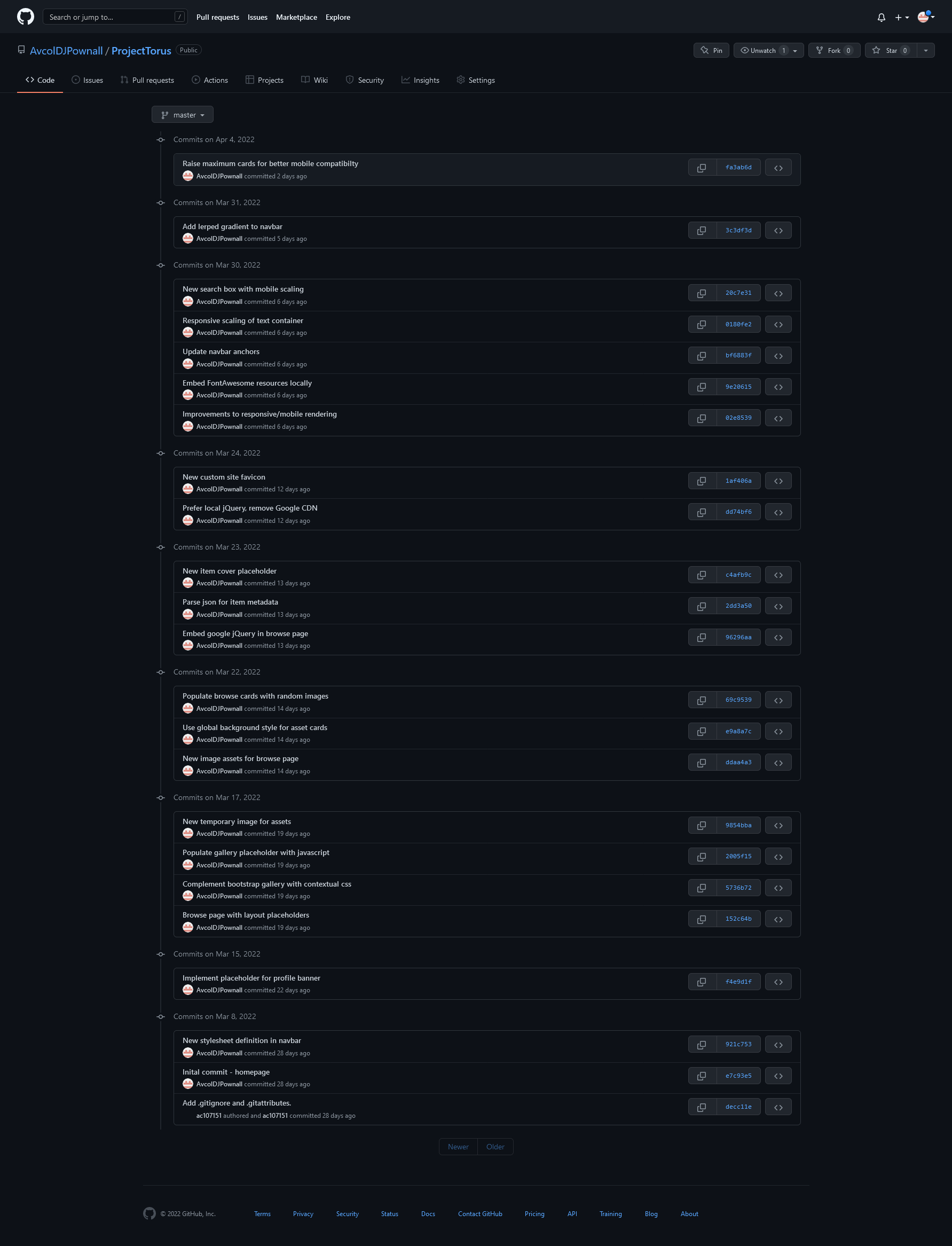


### GitHub Commits - Frontend

As with any long-term web project, responsible version management practices were required to assist the development of the site’s front-end. These changes were initially committed to the repository’s main branch, but have now been grandfathered under the ‘legacy-frontend’ branch to cleanly separate the project’s frontend and backend developmental stages. The site’s migration to from ASP.NET MVC 5.0.0 to 6.x will likely introduce breaking changes, as the [Semantic Versioning V2](https://semver.org/) standard indicates that API-incompatible changes were made in this ASP.NET version bump. Continuing backend development in another branch enables the website to be refactored with the latest version of ASP.NET MVC for the backend portion of development – ensuring the platform’s design remains as robust as possible.

The following GitHub link demonstrates changes to the site’s front-end throughout its development.

<https://github.com/AvcolDJPownall/ProjectTorus/tree/legacy-frontend>



# TORUS: Backend + Database

### Backend - Introduction

After the implementation of the Torus frontend, focus was shifted to the backend design of the website – including its database. While the Torus backend is built upon ASP.NET’s MVC-based platform, a large portion of the site’s design was prototypes using tools such as Microsoft SQL Server Management Studio.

This document explores numerous considerations, challenges and useful information pertaining to the design of the Torus backend.

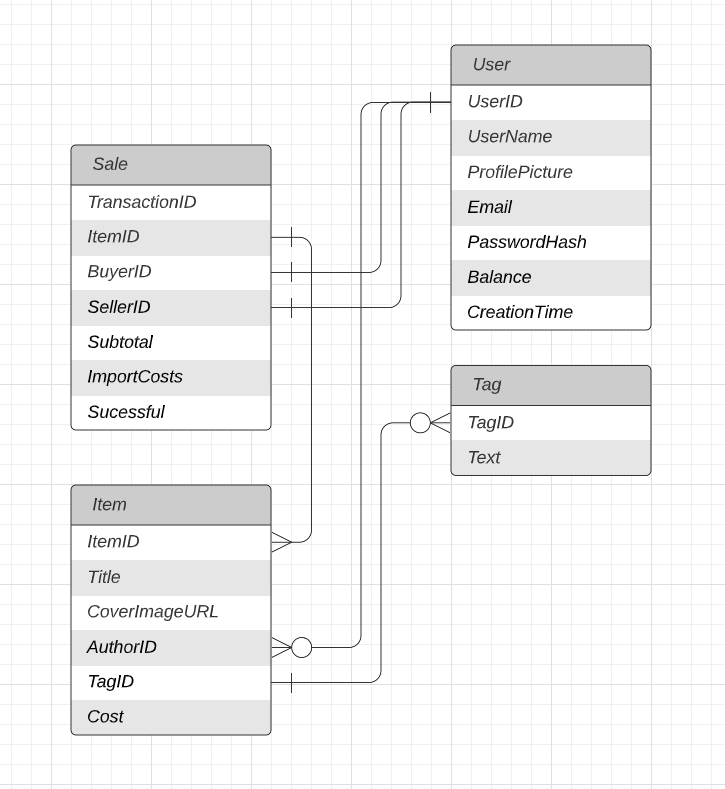
### Relevant Applications

Database security is a serious consideration in modern backend design. As such, a minimum privilege model has been employed where end users have limited control over database operations via a front-end panel. All user input is sanitized by ASP.NET’s input handlers, reducing the effectiveness of common attacks such as SQL injection. Other attacks such as cross-site scripting (XSS) will be mitigated through extensive testing of the frontend and backend components of the website. The end result of this testing should ensure that any HTML tags in user-submitted strings will be automatically filtered out. Another common attack that will be considered is cross-site request forgery (CSRF). To prevent other websites from embedding Torus pages to steal user credentials, the Torus project will be utilizing ASP.NET’s CSRF token implementation. This stores a temporary cookie in the user’s browser to validate that the user’s request is generated by an authenticated client.

User privacy has always been a defining aspect of the Torus platform – as reflected across its various stages of design. This is especially important in backend design, as users cannot directly verify how their data is processed on the site’s backend servers. As such, the Torus Project aims to offer a best-in-class solution to user privacy compared to competing services. While the site utilizes browser cookies for the purpose of authenticating the user between sessions, these are limited to the scope of the Torus webpage and cannot be used for cross-site tracking. The site’s user model stores an extremely minimal amount of sensitive information, all of which is used exclusively for operation of the site. This was done to mitigate the scope of a data breach, as retaining extra user data for the purposes of analytics was not worth the risk of exposing sensitive information to bad actors.

### Entity-relationship diagram

The following diagram represents the relationships between the site’s various tables. This database makes liberal use of linked tables which reference each other through numerical IDs. Efforts were made to make the design as intuitive as possible, as reflected by the cleanliness and minimalization of dependencies.

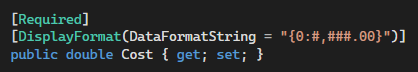


### Design + Development

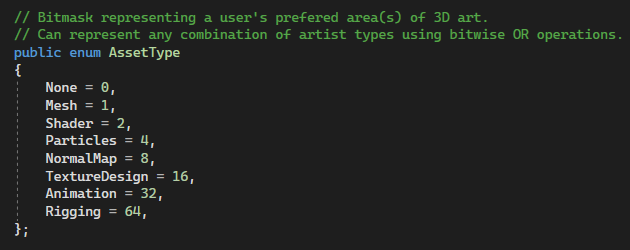
The Torus database was designed with the goal to make future maintenance as simple as possible. Such future-proofing would significantly reduce the workload of future maintainers, reducing the amount of time, money and resources required to keep the Torus project running. An example of such robust design is the decision to store the user’s funds as an unsigned 64-bit integer. Traditionally, decimal numbers are stored as floating-point numbers, where a mantissa is used to represent base 2 numbers similarly to scientific notation. However, this approach would come at the cost of reduced precision in floating point math. This can be an issue when floating point decimals are used to represent currency, where the subtle rounding errors directly affect the user’s displayed balance. The standard integer type also posed issues, as any number above 2^31 (2.14 billion) would result in an integer overflow. To remedy these issues, all balance and cost-related database fields are stored in cents through SQL server’s “big int” datatype, a 64-bit integer type which may store any value up to 2^63. All queries passed onto the frontend from the database would then be divided by 100, thus accurately providing the dollar cost as a decimal. In summary, this decision to represent all money-related values with a 64-bit integer greatly improves the database’s maintainability.



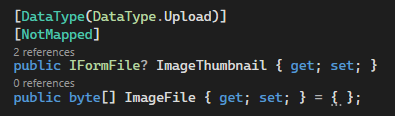
Moving to the MVC implementation, certain adaptations were required to fit ASP.NET’s type-based input validation. In short, a decimal value cannot be interpreted as an integer without first creating a dummy column. As storing an unused currency value in the database for the purposes of type handling would introduce needless complexity, the decision was made to represent monetary values as a floating-point decimal number in the TorusPosts table. While values such as the User’s balance must be represented as accurately as possible, the monetary cost of a single post is at no risk of encountering loss of floating-point precision given the platform’s range of valid prices. But for good practice, the cost of listed Torus posts will utilize a double-precision floating point number utilizing 52 bits to represent the mantissa rather than the 23 bits of precision offered by a single-precision float.



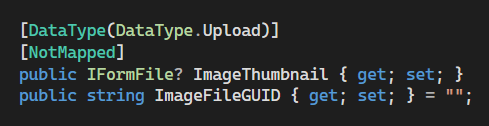
In Torus, any creator may specialize in a variety of fields surrounding 3D art. As such, no single category can truly represent the technical and artistic background of a given user or post. As such, a bitmask containing common fields of 3D art backgrounds was introduced to quickly and elegantly categorize the platform’s users and posts into combined areas of specialization. This system is used alongside the Tags table to categorize users with similar interests. Here, Torus uses the AssetType enumerator to encode any combination of 3D asset types. Due to the nature of bitwise operations, applying an OR operation between any two entries will result in one integer representing all relevant categories. This provides customers with a “specialization” ID, enabling users to identify others with similar backgrounds.



Later in the design process, the decision was made to upload thumbnails images directly to the site via a file upload prompt. This would allow Torus creators the freedom to create their own eye-catching preview image to express their digital product, greatly improving click conversion rates. However, this functionality introduces the challenge of storing images in a modern database. One potential storage method was to directly upload the thumbnail image to a byte array.



While this storage method proved successful, such an approach would not be an ideal long-term solution for handling storage of file uploads. Each time an image is displayed on a webpage, the server would need to request and deserialize a large file directly from the database, increasing render time and introducing unnecessary load on the server. Instead, the decision was made to only store a unique reference to the image file in the database rather than the image itself.

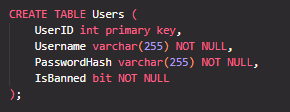




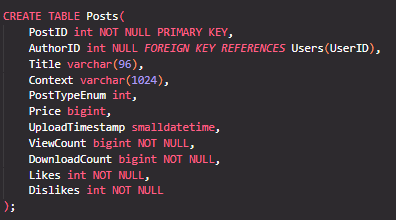
The above upload handler generates a unique file identifier for each uploaded thumbnail, then writes the resulting file into the ~/img/items directory. This defers the role of image storage to the webserver itself, rather than the database. As most webservers such as Apache2, HTTPD and Azure utilize a webserver’s file system for content delivery, this approach makes the most sense in modern webhosting. Each time the Torus webpage requests a preview image, the database only needs to provide a 16-byte reference to the image – the same unique identifier corresponding to the uploaded file name. This allows modern web caching and CDN optimizations to further improve server-side performance through ensuring that a static file is requested from the webserver.

### Table Structure

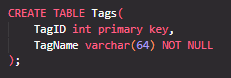
The first table created in the Torus project is the Users table. As demonstrated below, a minimal amount of user data is stored by the Torus project. This is an intentional factor in Torus’s design, as one of the platform’s core attractions is its high focus on user privacy compared to other figures in the industry.



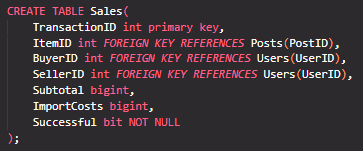
The Posts table contains by far the most metadata in the Torus database. This greatly improves the user’s ability to search for relevant 3D assets, thus allowing sellers to more easily monetize their work. Likes, dislikes and user interaction statistics are incrementally counted to allow users to browse the platform’s most enjoyed content.



Similarly, the Tags table can be utilized by Torus sellers to make their products easier for customers to find. The table contains a short string containing up to 64 characters of text for the purposes of making searches easier for users. Providing users with tag-based SEO tools greatly improves the user experience (UX) of the website’s search functionality, whist boosting the seller’s – and platform’s – income.

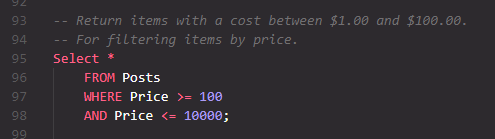


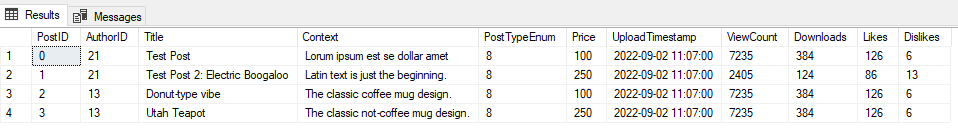
The Sales table is crucial for documenting the platform’s finances, while providing users with the option to save receipts for their orders. The table relies on three foreign keys, two referencing the Users table and one referencing the Posts tables.



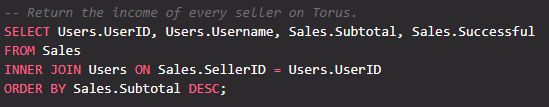
### SQL Queries

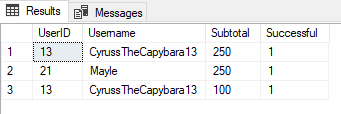
In a marketplace such as Torus, users will typically have a preferred budget range for buying digital goods. As such, functionality for filtering by budget is a good quality-of-life choice for consumers. The query below returns all posts on the platform between 100 cents ($1.00) and 10000 cents ($100.0).



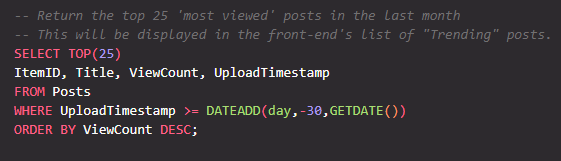


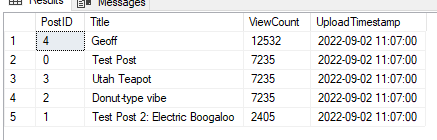
Knowing which sellers on Torus provide the highest income to the platform is crucial for improving services and communication to the most valuable user-generated content producers. The following query reports the reported income of all active sellers based on the subtotal of their previous sales.



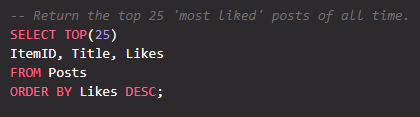


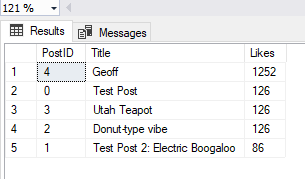
To better promote the platform, users of Torus should be encouraged to make engaging content which catches the eye of new visitors. The following query returns the most viewed items in a given month, encouraging active participation in the Torus community.





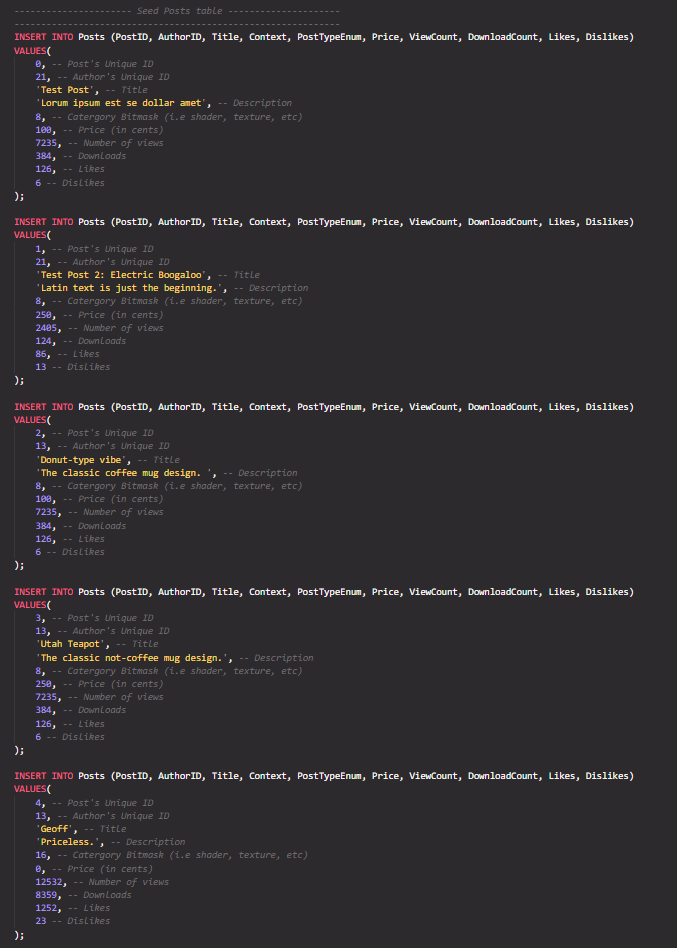
The same applies to keeping track of the platform’s overall most-liked posts. Providing a section for filtering by the most user likes helps users discover the absolute best creative output of Torus users.





### Seeding The database

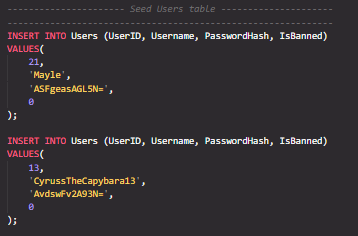
The Posts table is by far the most important source of data in the Torus platform. As the platform is a large repository of user-submitted content, plenty of columns are needed for designing and testing SQL queries.



The Tags table is another important data source, as this greatly improves the end user’s ability to filter through posts. Below, five example tags have been generated.



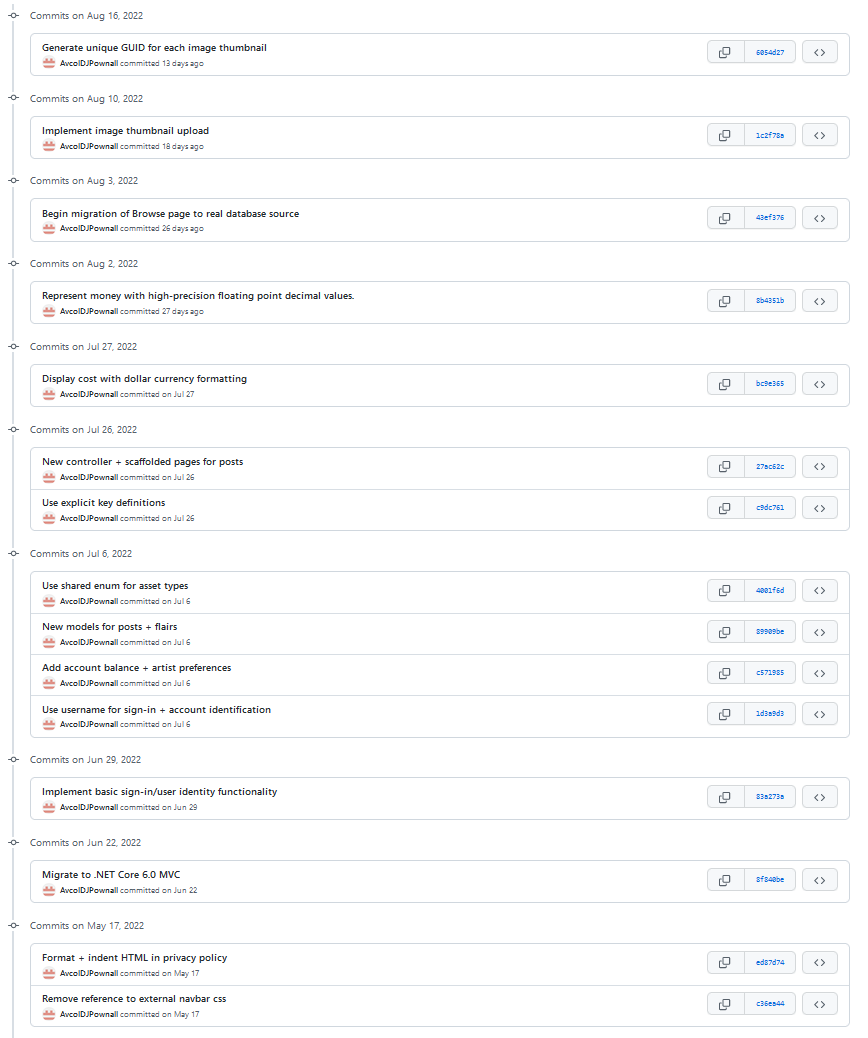
Finally, these are the two seeded entries representing Torus users. As the example queries in Torus don’t particularly require a large dataset of users, only two have been seeded for testing purposes. This is sufficient for most Alice and Bob style user interactions.



### Github Commits - Backend

Maintaining good version control practices is of upmost importance in the world of database design. This is no different for the Torus backend, where a new branch in the ProjectTorus repository was created to back the project’s migration to ASP.NET’s MVC framework. An extensive collection of GitHub commits document the backend design across various stages of its development, as demonstrated below.

<https://github.com/AvcolDJPownall/ProjectTorus/tree/torus-backend>



### Maintenance

The Torus front-end was designed with future functionality in mind. While full functionality of the Torus marketplace was beyond the scope of this initial front-end design, the project’s UX development aimed to make adapting an appropriate backend significantly easier.

The included placeholder item metadata is formatted in a similar fashion to a real backend, remaining modular for future maintainers to easily implement their own backend using ASP.NET’s razor pages.

The web application’s simplistic design aimed to keep core functionality easier to maintain. The existing front-end offers the key components seen in online marketplace websites (browsing areas, search filters, etc.) with little unneeded functionality. This makes the website more flexible for future maintainers to add their own necessary feature set for Torus, saving them from dealing with the unneeded bloat.