Adeptus Astartes Web Application for

Master of Science in Information Technology

Software Design and Programming

James McKenna and Estell Moore

University of Denver University College

9/27/2024

Faculty: Greg O'Toole, PhD

Director: Gregory Reese, PhD

Dean: Michael J. McGuire, MLS

Abstract

One of the most popular franchises that has only gained more traction and publicity over the last 40 years has been Warhammer 40k. This franchise has garnered a lot of attraction and popularity with its expansive fictional universe and lore. The problem is a lot of this information has either been retconned or completely changed due to new books, new factions, and new ideas that Games Workshop has come out with. The current wiki for Warhammer 40k is disorganized, and heavily influenced by users directly editing pages without fact checking against the sources. The idea of this web application is to provide easy access to fact-checked source material, and will give users not only information they seek in a tidy and easy to use web app, but also allow users to experience a social Warhammer setting.

TABLE OF CONTENTS

Components

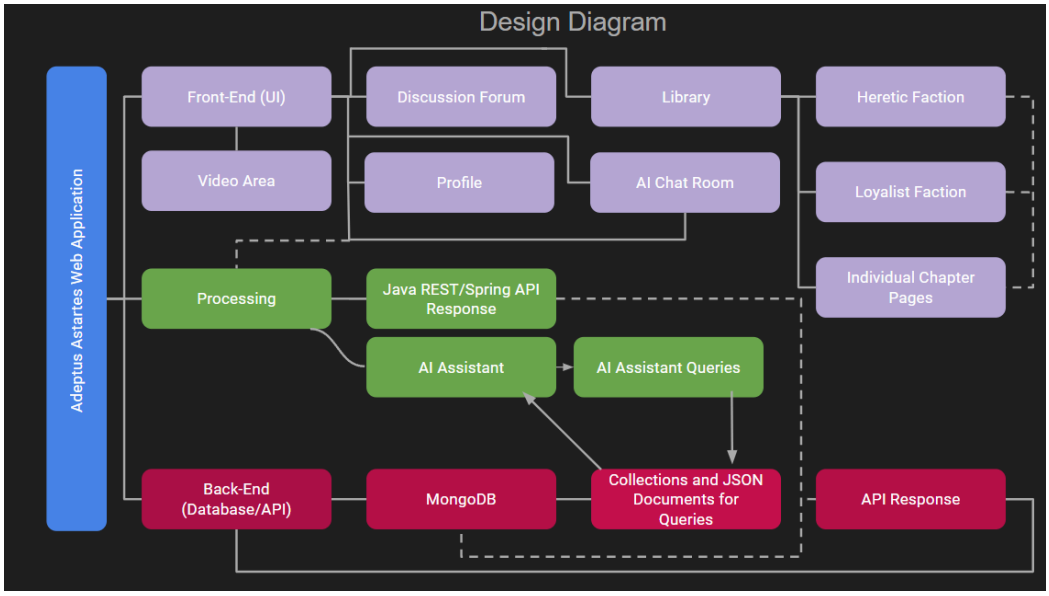

Figure 1. Design Diagram for Adeptus Astartes Web Application

Figure 1 outlines overall flow and design summary of the process flow for the web application. This is broken up into Front-End (UI), Processing, and Back-End. The Front-End of the application will be extremely involved–primarily following structured UI Layouts to help with planning. To discuss the tiers of the components in the application, it has been split into different features. With the description of these features, there is a detailed description of how front-end, back-end, and processing occurs in all.

The Warhammer Library

In Warhammer 40k, there is a substantial amount of lore that has spent decades in constant development. The main goal of this web application would be the organized library that holds all relevant lore for the Space Marine Chapters. A fun additive to the library would be a separation between two factions: Loyalists versus Heretics. For the application, when users

are searching certain areas, they will be separated between the two factions. Also, the library will primarily contain information solely to Adeptus Astartes also known as Space Marines.

Front-End

This directly affects the front-end UI, where colors and stylesheets will update on a page based on the user's selection of which faction they are researching to reflect the lore of Warhammer. If the user selects Heretic, the stylesheet will have black and yellow colors, and the styles of borders and fonts will be sharp and spiked. The intention would be for the UI to be more chaotic, yet still functional. If the user selects Loyalist, the stylesheet will have blue, green, and red colors with stylistic borders and fonts being more rounded. The intention behind this design would be structural–and a fun experience for the user. For both factions, the general UI layout would be a Catalog Layout with displays for each relevant Chapter for the factions (Shaikh, Roshni. 2023.).

Under each faction would be a Hierarchical UI Layout for each Chapter under the user's selection. The Hierarchical layout would connect the faction with an image of the Chapters, the title, and a link to send the user to the Chapter's sub-library. The sub-library's would then follow a Multi-Panel UI Layout with everything relevant to the user's selected Chapter, as well as change its CSS based on the Chapter's colors.

Processing

Processing will handle whether the user is on the heretic homepage or loyalist homepage. Depending on the page selected by the user is what default Chapter information it

will process to the user, and communicate with the back-end DB to display to the UI. This would be handled through Java and Rest API's to process this data, and then will be communicated with the database to retrieve the relevant information. This information is then returned to the UI to change the stylesheet and information the user sees. Each library page would have its own process to display different layouts and styles depending on what the user selects.

Back-End

The database needs the factions separated into two separate collections–loyalist and heretic. Ideally, the MongoDB database would contain core collections and documents (json-like data) that will pass back information to the processor for the UI. This will allow for the overly complex nature of various relationships, events, and lore that each Space Marine Chapter has tied to it. Also this ties into the API Response to talk back to the front-end part of the website so the user can receive the information that they have requested. It will contain each Space Marine Legion, their Chapter, their Primarch, key events and battles, and other deep lore that is specifically tied to that Space Marine Chapter.

User Profiles

Because this is an interactive community, the option for allowing users to create customizable profiles feels like one of the most integral features of the web application.

Front-End

Users can set their profile based on their favorite faction, whether it's loyalist or heretic. Users will have an option to select their faction, and will be displayed an interactive widget at the top of the profile to redirect users to the relevant faction library page. Next to the insignia will be the user's profile picture. Then, users can stylize their profile by selecting a Chapter from the chosen faction, which will update the CSS for the page to reflect their selection. Ideally, the layout of the profile would be a Tab UI Layout where one tab is an "About Me" tab to showcase the users interests, details, and biography, a "Posts" tab that shows the users history of posts and responses from discussions, as well as homebrew Chapters they create, and the final tab would be a "Video" tab that will display the users liked videos, uploaded videos, and live streams (Shaikh, Roshni. 2023.). Then, the user will be allowed to add or subtract shelf widgets inside each tab to display different categories of what they would like to showcase on their profile.

Processing

In the Customizing screen of the profile, the processor will retrieve the Chapters based on if a user selects "heretic" or "loyalist". The selections will communicate back to the UI whether or not a CSS template is available for a faction. Then, the processor will retrieve the template profile from the database, organize the endpoints and tags, and send it to the UI to update it. Also an important factor, when users navigate to a profile, this data needs to be saved. The processor can process the saved data for the customized profile and communicate the data to the UI to swap the appearance.

Back-End

A Core Collection would be created, with a Document within it, that would pick and choose all the colors of known Space Marine Chapters. This would then be sorted and stored for the UI/UX to parse (MongoDB. 2024). A colors field would be added into the document so that users would be able to choose between a primary and secondary color for their chapter; as they would see it on the UI/UX. The chapter the user comes up with would then store and save this data into their user profile, which would be stored within a section of the database; primarily for user profiles.

Discussion Forums

An area of the web application will be a community discussion forum where topics, categories, and ideas could openly be discussed among users. It will similarly follow a social media format, but geared more towards organized ideas instead of a timeline of information.

Front-End

The forums will be a separate section of the application aside from the library and profiles. This area will follow a Timeline UI Layout that displays posts made by users in the application (Shaikh, Roshni. 2023.). This can be filtered by Most Popular or Most Recent. There will be sidebars for navigating categories, a search bar at the top for easy access to search for keywords, and the scroll behavior would follow an Infinite UI Layout. This would allow users to infinitely scroll down the timeline of posts by users if they choose not to filter or search results.

The posts will be displayed on the main and category page of the Discussions in a Card-Based UI Layout (Shaikh, Roshni. 2023.). It will display limited information about a post, simply the profile picture of the author, the title, and a short excerpt from the discussion. Then, each post sub-page would display the name of the author, their profile picture, the title, and the discussion area along with its upvotes/downvotes. At the bottom of the page would be an area for other users to leave comments on the post, thus resulting in the discussion.

Lastly, the UI will provide a templated widget for a user to submit a discussion. This would consist of different text fields they can fill out or edit.

Processing

The processor needs to retrieve post data from the database when the user loads the discussions page. The data it loads could possibly be stored in the cache of what the user has filtered on in the past for the main page, and display to the user the relevant posts. For the “Most Popular” filter, there could be a field included in the database that the processor could organize in ascending to descending order, and display appropriately to the user.

When the user creates their own discussion, the processor would automatically process the author’s name, profile picture, and dates/times for the post. Then, it would retrieve each text field’s values, organize those values, and send them to the database to be stored. Different areas of the application will be updated as well–for example, if a user submits a new post, their post should show up with the “Most Recent” filter, or update with possible tags they could add to their post. This would affect other users and their searches.

Back-End

Have a json style document in MongoDB dedicated to discussions, each record is separated into its own document within MongoDB. The document would be stored with information based on the user, title, discussion and tag to be retrieved and displayed to the UI. All discussions would also be stored into a core collection with MongoDB (MongoDB. 2024).

Warhammer AI Assistant

A feature of the application that could set it apart from other references would be the development and interaction with an AI Assistant that specifically is designed with Wahammer 40k lore.

Front-End

The application will have a chat room that is displayed in a Stacked UI Layout (Shaikh, Roshni . 2023.). It will be similar to a text messenger format where the AI will respond in a message bubble, and the user will respond in a text chat beneath, and so on. This AI's appearance would have a designated, fun profile picture of an Adeptus Mechanicus character from Warhammer, to simulate as if the user is speaking with one of these characters.

Processing

The processing will involve intaking the user's message sent to the UI, and passing it into the Machine Learning model. The back-end will determine how to respond to the user, and the processor will pass this response in a chat bubble display back to the user. The messages the

user sends could also be filtered in the processor with automated messages depending if the user sent an empty bubble, or repeated their question. It could cut down the back-end processing time for the ML model to respond if it's simple complications.

Back-End

For the second tool on the back-end we'll talk about NLU paired with RLU. A Natural Learning Language will be the core machine learning model with all of its algorithms tied into it. The models that are primarily used for this would be the Open GPT model or a similar model (OpenAI. 2024). These can be fine-tuned with various parameters that would allow for user queries to dive into the MongoDB database that is set up for the use case. With GPT-like models for Space Marine lore, you can fine tune this with various data such as descriptions of the chapters, events, battles, legions, relationships etc. That would then fine tune the model to pull more useful data for future queries so that every new query returns more useful data to the front-end. Reinforcement Learning also builds off this as it uses the NLU model but it slowly helps train that model over time (Mnih, Volodymyr 2015). So the first few queries a user puts in can be completely wrong. However with a user feedback loop, asking if the results were desirable, this would train the RLU that would get better every single time with its queries (Stable-Baselines3. 2024).

User-Made Chapters (Wiki)

Part of the fun of the existing Warhammer 40k Wiki is that users can contribute to information about Chapters. This application will still have the capability for the community to

come together to create their own user pages regarding the Chapters, and will be entirely separated from source material, as well as allow users to create their own and be creative!

Front-End

This UI will be very similar to the source Warhammer Library discussed earlier in this document. It will be separated into the relevant factions, following the CSS from the Library, but there will be a slight twist. Instead of it being a source-material library, it will be a community wiki library to alphabetically display user-made references in a Category UI Layout. The pages will follow a basic wiki layout with images, descriptions, bulleted details, and a gallery. When a user wants to create their own wiki page, the UI will display a template the user can fill out text fields with. They can also add/subtract widgets from the template to customize their wiki page however they please, and allow more experienced users to utilize CSS on their page.

Processing

The processor will function very similarly, if not identically, to the Warhammer Library discussed earlier in this document. The only difference is the processor will save off the users creations, similar to the same model that the Discussion Forums will follow.

Back-End

From a back-end standpoint. There will most likely be another core collection and document attached to it, solely focused on user made chapters. This might also be tied in with the user profile; so the collection in MongoDB might be stored under user profiles potentially.

To wrap it to the front-end, when the user filters for community-based entries, there will likely be a field that indicates whether it's user-generated. This would help with the search engine on the front-end once the processor sends the query for what the user is looking for. When users save their entry, the data will be saved to the database, and stored for the filter to process through once communicated by the processor.

Video References

The application will have a function where users can find Warhammer videos through the YouTube API, save the videos, upload their own custom Warhammer videos, and possibly LIVE stream videos to the platform.

Front-End

The video section of the web app will be a separate landing page, but the library will have many video references on the relevant pages to browse relevant videos. It will follow a combination of Infinite Scroll layout and a Card-Based Layout (Shaikh, Roshni . 2023.). Similar to YouTube's layout, there will be a thumbnail of a video, and to the right of it, there will be its title and description. The most important part is the UI needs to display YouTube videos separately from videos that user's upload to the web application under their own profiles.

Also, on user profiles, the video tab must be updated with their saved/liked videos, as well as personally updated videos. When users want to content-create or upload their own videos, there will be a templatized upload section, possibly a "Call to Action" (CTA) area, where the user can upload their video, add title and description to the video, add relevant tags and

links (Brinker, Mark. 2019.). These are simple text fields, possibly also including some drop-downs for genre selections for their video as well.

On the separate page of videos themselves, they will have a right-side scrollbar of recommended videos, which the processor will handle how to display these. The display will be titles and thumbnails of the videos, and preferably a darker CSS to give a movie-theater feel to the video section.

Processing

Not only will the processor have to handle how to display the default recommended videos on the video home page, but also handle all recommendations for users. There will need to be a cache of the user's history, possibly handled by back-end tags the videos will have. Depending on the cache is which videos the processor chooses to recommend to the user, as well as display on the home page. If there is no history from a user, the videos will be automatically filtered based on a mix of "Most Popular" and "Most Recent".

Another aspect the processor will have to handle is the actual uploading of videos themselves. From the CTA area for uploading videos, the processor must intake all the text fields of information, as well as retrieve and compress the video to send to the database/server. Essentially, the processor here is most important to organizing, containerizing, and saving off the information of the video.

Back-End

The back-end of this area will be a bit more complicated, as it's now having to handle the saving of videos. The application will need to have a server to handle the request, and though MongoDB can handle storing of videos, there may be other options to research that could handle it easier. MongoDB will store the video in a binary format (BSON), and can handle up to 100MB videos–but if the user uploads a video that's larger than the allotted size, there will have to be an error thrown to the user that the video file is too large (Alger, Ken W. 2022.). For the purpose of the application, on a small scale, MongoDB will suffice.

Also, for the recommended YouTube videos to be displayed to the website, the processor will need to communicate with a YouTube API. There is a Java API available for YouTube, and this can be utilized to bring over Warhammer-related videos to the web application. Although, specifically, it would need a strict filter for *only* Warhammer videos so that the application doesn't gear towards something it's not intended for.

Domains

The application will be supported on a few different devices and domains. Primarily, it will be supported as a Web, PC/Mac web application. The decision behind having it on Desktop as a web app is because Warhammer games are mainly on PC/Mac. It'd be nice to have because most people that regularly play video games spend a lot of time on their PC, so the web app could be easy-access to referencing source material from video games, or even a form of social media with like-minded people that is accessible via a computer, whether it be the web app or website.

An IOS/Android mobile device application would also be made available, considering certain fans of Warhammer prefer to read books or play a table-top game of it. This would allow this application to be versatile and act as a handy fact checker. It'll be a more limited mobile app, not as verbose as the desktop version, but something mainly utilized as an immediate reference from the library.

Tools

There will be quite a few tools this application will need to utilize. Some tools won't be directly expanded on, as some are simply just languages, but the list of which languages will be used will be combined as a brief explanation.

Figma

For planning and UX Design, it'd be best to utilize a tool such as Figma. Figma is a powerful tool used for planning in a collaborative team setting, and allows teams to also generate mock-ups for UI for web and mobile applications. This would be all front-end work, mapping out landing pages, home pages, and child pages of the website, but ensuring the applications are overall intuitive and easy to use. Since this application would be developed by multiple users, it would also prove to be a great collaborative space for the needs of the team.

HTML/CSS/JavaScript

The main languages for the front-end development will be HTML (HyperText Markup Language), CSS (Cascading Style Sheets), and JavaScript. The pages of the web application will be laid out with HTML, they will be styled using CSS, and then processed using JavaScript. HTML

is best utilized for structures of web pages and applications. It handles the visual content of a page, whether that be setting up paragraphs, tables, images, or widgets (MozDevNet. 2024.), CSS will handle the style and layout of a web page. It is best used for colors, stylesheets, fonts, borders, and spacing on a page (MozDevNet. 2024a.). The most powerful of these languages and best used for dynamic manipulation of HTML and CSS for scalability purposes will be JavaScript. JavaScript is best used for many web-based development, mainly to update HTML/CSS elements, validate forms, and make asynchronous requests to the server (Simplilearn. 2024.).

Java/REST API/Spring

For back-end processing, Java, REST API, and Spring will likely be used. REST API (Representational State Transfer) is mainly utilized for designing network applications that follow a client-server model that is driven by resources called endpoints. REST is geared heavily towards scalability, offering stateless communication, client-server separation, and resource-based interactions (Doglio, Fernando. 2023.).

Spring Framework is an open-source Java platform which is basically a “dependency injection container” (Behler, Marco. 2022.). It’s best used as a model for Java enterprise applications, ensuring that these applications won’t have to handle working with data source dependencies (Spring Framework. 2024.). This specifically helps with connections to databases and cloud-based data services. Though the infant stages of the application may not be via the cloud, it’s something on the roadmap of improved development.

Java is the main back-end language that will be utilized to interact with REST and Spring–as well as communicating to the databases via XML (Extensible Markup Language) or JSON (JavaScript Object Notation). Java is an object-oriented programming language specifically created to handle the building of web applications, and there are many resources online in the form of API's to work with Java specifically.

GitHub/Git

It's important to address the team development environment for creating this application. This is where Git and GitHub can be best utilized for version control, as well as preventative measures for conflicts in code development. Git is a version control system where teams of programmers can collaboratively update and create code simultaneously (Git. 2024.). With the use of Git, the team would also need to utilize some sort of platform that could handle the ability to collaborate and access the same repositories/code base. This can be done by utilizing GitHub. This is a cloud-based platform that utilizes Git, and allows developers to collaborate on the same projects (GitHub. 2024.).

MongoDB

For the database tool we'll talk about MongoDB and the particular reasons as to why it's being chosen. The reason we picked MongoDB as a NoSQL or non-relational database, was because it's flexible, it allows for unstructured or semi-structured data. This will work well with parsing books and other various forms of content that will be stored into the database (MongoDB. 2024). It also allows for the developer to not tie into a schema like MySQL that is

strictly, key-value, or row and column based. This will hinder various sorts of data from being stored automatically. MongoDB also is stored based off of Collections and Documents. Core collections are categorized documents with documents being a JSON-like data structure (MongoDB. 2024). The typical Core Collections that could store data would look something like this:

1. Core Collections:
    - Legions (Loyalist, Heretic, or Renegade)
    - Chapters (linked to legions)
    - Battles
    - Primarchs (The leaders of the Space Marine Chapters)
    - Lore Entries (Specific lore that is tied to only that Space Marine Chapter/Chapters)
2. Document Examples (JSON Format):
    - See Appendix A-1 for Code Examples

Maintenance Plans

For Updates and Maintenance, ideally this would be developed by a Scrum Team. This would allow for updates, features, scalability and maintenance as a team would handle and manage this piece of software. A team would primarily be designed with Fullstack Engineers to handle the front-end and back-end, a Product Owner who would own and prioritize the backlog of work, alongside talking with respective stakeholders and other users. A Scrum Master who

would handle the Scrum Ceremonies and work flow for the team. There would be a Software Architect who would instruct on proper software techniques and design patterns for the team to follow. Lastly, Database Engineers and Machine Learning Engineers would be used to help maintain the artificial intelligence and database that occurs in the background of the web application. ALso, some of the selected tools, like REST API and JavaScript, are also selected to ensure good scalability for the application.

References

Alger, Ken W. 2022. “Storing Large Objects and Files in Mongodb.” *MongoDB*. MongoDB Inc. May 13. https://www.mongodb.com/developer/products/mongodb/storing-large-objects-and-files/.

Behler, Marco. 2022. “What Is Spring Framework? An Unorthodox Guide.” *Learn More about Java, No Matter Your Skill Level, Anytime and Anywhere You Want - Marco Behler GmbH*. Marco Behler GmbH. June 23. https://www.marcobehler.com/guides/spring-framework.

Brinker, Mark. 2019. “Parts of a Website: A Cheatsheet for Non-Techies - Mark Brinker.” January 24, 2019. https://www.markbrinker.com/parts-of-a-website.

Doglio, Fernando. 2023. “What Is REST API in Java? Guide with Examples.” *Camunda*. September 5. https://camunda.com/blog/2023/09/what-is-rest-api-in-java-guide-with-examples/.

Git. 2024. “1.3 Getting Started - What Is Git?” *Git*. Accessed September 29. https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F.

GitHub. 2024. “About Github and Git.” *GitHub Docs*. Accessed September 29. https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

Google. 2024. “Dialogflow Documentation | Google Cloud.” *Google*. Accessed September 29. https://cloud.google.com/dialogflow/docs.

MongoDB. 2024. "Aggregation Framework." MongoDB Documentation. https://www.mongodb.com/docs/manual/aggregation/.

MongoDB. 2024. "Data Modeling Introduction." MongoDB Documentation. https://www.mongodb.com/docs/manual/core/data-modeling-introduction/.

MongoDB. 2024. "Model Tree Structures with Ancestors Array." MongoDB Documentation. https://www.mongodb.com/docs/manual/tutorial/model-tree-structures-with-ancestors-array/.

MozDevNet. 2024. “HTML Basics - Learn Web Development: MDN.” *MDN Web Docs*. Accessed September 29. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics.

MozDevNet. 2024a. “CSS: Cascading Style Sheets: MDN.” *MDN Web Docs*. Accessed September 29. https://developer.mozilla.org/en-US/docs/Web/CSS.

OpenAI. 2024. "GPT-3 and GPT-4 Documentation." OpenAI Documentation. https://platform.openai.com/docs/.

OpenAI. 2024. "OpenAI Gym Documentation." OpenAI Documentation. https://gym.openai.com/docs/.

Ray. 2024. "Ray RLlib Documentation." Ray Documentation. https://docs.ray.io/en/latest/rllib.html.

Shaikh, Roshni . 2023. Review of 30 User Interface Layouts Used in UI Design. DevSquad. DevSquad. June 12, 2023. https://devsquad.com/blog/user-interface-layouts.

Simplilearn. 2024. “10 Practical Applications of JavaScript and Tips: Simplilearn.” Simplilearn.Com. Simplilearn. August 13. https://www.simplilearn.com/applications-of-javascript-article.

Smith, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Tech. *Playing Atari with Deep Reinforcement Learning*. Toronto: DeepMind Technologies.

Spring Framework. 2024. “Spring Framework”. *Spring Framework*. Accessed September 29. https://spring.io/projects/spring-framework.

Stable-Baselines3. 2024. "Stable-Baselines3 Documentation." GitHub. https://stable-baselines3.readthedocs.io/en/master/.

Warhammer, n.d. “Warhammer 40,000.” *Warhammer 40,000*. https://warhammer40000.com/

Appendix

A-1: Code Examples, MongoDB:

Loyalist Example:

```
{
  "legion_name": "Ultramarines",
  "allegiance": "Loyalist",
  "primarch": {
    "name": "Roboute Guilliman",
    "status": "Living",
    "homeworld": "Macragge"
  },
  "chapters": [
    {
      "chapter_name": "Ultramarines",
      "founding": "First Founding",
      "chapter_master": "Marneus Calgar",
      "homeworld": "Macragge",
      "battle_cry": "Courage and Honour!",
      "colors": "Primarily Color is Blue and Secondary Color is Yellow",
      "notable_battles": [
        {
          "name": "The Battle of Macragge",
          "date": "745.M41",
          "enemy": "Tyranids"
        },
        {
          "name": "Siege of Terra",
          "date": "014.M31",
          "enemy": "Forces of Horus"
        }
      ]
    },
    {
      "chapter_name": "Novamarines",
      "founding": "Second Founding",
      "chapter_master": "Tiberius Silas",
      "homeworld": "Honourum",
      "battle_cry": "Courage and Honour!",
      "notable_battles": [
        {
          "name": "Defense of Honourum",
          "date": "734.M41",
          "enemy": "Orks"
        }
      ]
    }
  ]
}
```

Heretic Example:


```
{
  "legion_name": "Word Bearers",
  "allegiance": "Heretic",
  "primarch": {
    "name": "Lorgar Aurelian",
    "status": "Daemon Prince",
    "homeworld": "Colchis"
  },
  "chapters": [
    {
      "chapter_name": "The Host of Lorgar",
      "founding": "Traitor Legion",
      "chapter_master": "Kor Phaeron",
      "homeworld": "Sicarus",
      "battle_cry": "The Truth Revealed!",
      "colors": "Primary Color is Red, Secondary Color is Black",
      "notable_battles": [
        {
          "name": "Battle of Calth",
          "date": "005.M31",
          "enemy": "Ultramarines"
        }
      ]
    },
    {
      "chapter_name": "The Serrated Sun",
      "founding": "Traitor Legion",
      "chapter_master": "Zardu Layak",
      "homeworld": "Sicarus",
      "notable_battles": [
        {
          "name": "The Shadow Crusade",
          "date": "008.M31",
          "enemy": "Imperium of Man"
        }
      ]
    }
  ]
}
```


1. Example Usage:
   Querying for Loyalist or Heretic Chapters:

```
// Find all Loyalist chapters
db.legions.aggregate([
  { $match: { "loyalty": "Loyalist" } },
  {
    $lookup: {
      from: "chapters",
      localField: "_id",
      foreignField: "legion_id",
      as: "chapters"
    }
  }
])
```