Computer Science
COC251
B927949

Creating a knowledge base for The Binding of Isaac

by

Tyler J. Bowcock

Supervisor: Dr. D D Freydenberger

Department of Computer Science
Loughborough University

May 2023

Abstract

Contents

List of Figures

List of Acronyms

ACID Atomic, Consistent, Isolated, Durable

AWS Amazon Web Services

HTML Hyper-Text Markup Language

HTTP Hyper-Text Transfer Protocol

JSX JavaScript XML

SQL Structured Query Language

WSGI Web Server Gateway Interface

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Problem Definition

Item interactions are an important mechanic of most modern roguelike/roguelite games, including The Binding of Isaac. However, with hundreds of items, each with a handful of good or bad interactions, it is nearly impossible to effectively remember them all. Graph databases are purpose-built to store and navigate relationships.[1] The output of this project will be a web application that leverages this feature of graph databases to allow users to query item interactions in The Binding of Isaac.

1.2 Aims and Objectives

The goal of this project is to make querying item interactions in The Binding of Isaac quicker and easier by using graph databases. Users will also be able to update the data in the database to ensure it matches any changes in the game.

The aims of the project are to:

1. Create a graph database containing relevant data about The Binding of Isaac.
2. Develop a web application that utilises a graph database to helps users to find item interactions in the game.
3. Explore testing methodologies to aid in producing a stable application with high quality code.
4. Search for possible ways to extend the project with future updates.

1.3 Risks and Constraints

Cost

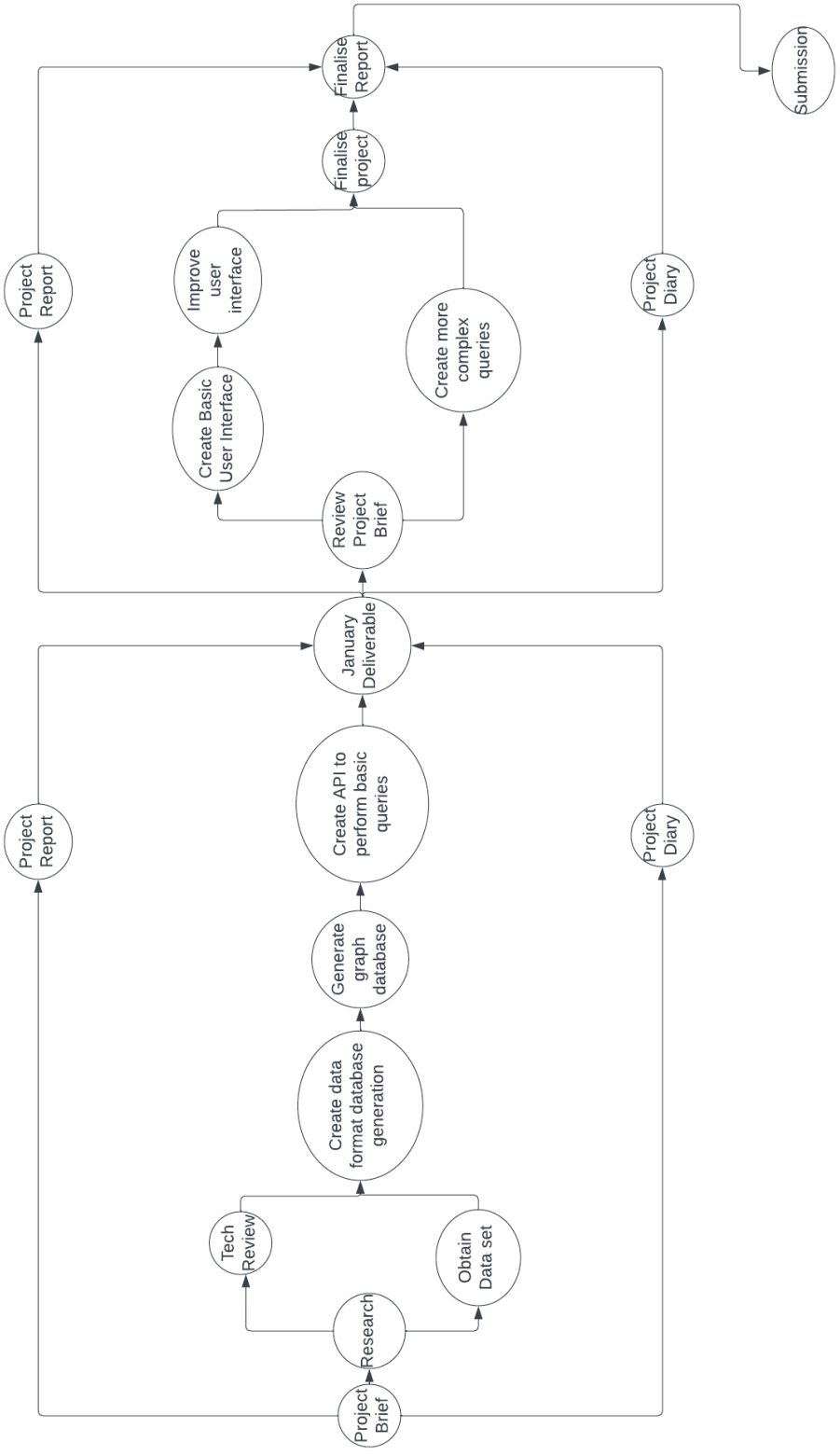
This project has no budget and so any services used in the development of the application will need to be free.

Dataset Availability

The data needed to create the database may become unavailable or unusable.

1.4 Project Plan

Figure 1.1: PERT chart showing project flow



Chapter 2

Background

2.1 Introduction

Through this chapter research will be conducted into varying areas related to the project. The conclusions made from this research will support any design or implementation decisions made in the course of the project.

2.2 The Binding of Isaac

'The Binding of Isaac: Rebirth is a randomly generated action RPG shooter with heavy Rogue-like elements.'[2] The player progresses through 'floors' which are made up of a series of 'rooms', each room can contain a variety of enemies, traps, and items. Each floor has a set of 'special rooms', namely a boss room, item room and a shop. The goal is to use the items found on each floor to defeat each boss, progressing to the next floor until the game is finished. The items in the game often interact, this interaction is usually called a 'synergy' if it benefits the player. In this instance the rogue-like elements are that the game has to be successfully completed many times, these are called 'runs'. Each run is unique and depending on the actions the player takes in the run it can have different outcomes and more parts of the game can be unlocked.

2.3 Existing Solutions

While there is no existing solution that is a direct comparison to this project, there are applications that have a similar purpose. These will be analysed to determine what features should also be implemented in this project and what could be improved by this project.

Fandom Wiki

screensho

The Binding of Isaac has two wiki sites hosted on the Fandom Wiki platform; one for the original flash game[3], and one for the modern version, commonly referred to as 'Rebirth'[4]. For the purposes of this project we will only be considering the modern version as it is widely considered the 'goto' version within the game's community.

The website contains comprehensive information on all aspects of the game, and it is continually updated by the community. Users can navigate the site using either predefined categories or a powerful search tool.

Advantages

- Contains information on all aspects of the game
- Actively maintained by the community
- Useful search functionality

Disadvantages

- So much information can make it hard to find what is relevant
- Unable to search for interactions, have to go via each item

Platinum God

screensho

Platinum God is a self-described 'Isaac Cheat Sheet'[5], and it contains item and key mechanic information for all versions of the game. The site is maintained by one person, and it claims to be more accurate than the community wiki as its updates are 'tested thoroughly in the game using Cheat Engine'[6]. The information is split into pages based on the version of the game; users can navigate this using the item icons which are arrayed on the page, or by using the search functionality. The search tool has some supported keywords, but will still usually require entering an exact match to an entry in the data. For certain versions of the game there is also a synergy finder tool which lets the user enter two items to see how they interact. However, this is limited to older versions of the game and only a small set of the items are actually included in the tool.

Advantages

- Information is more reliable than the community wiki

- Easier to reference quickly due to there being less information

Disadvantages

- Only one maintainer can mean long update times
- Only contains basic information about each item
- Limited or no synergy information for most items
- Harder to find items without knowing the name or what the item looks like

2.4 Technology Review

This section will research the potential technologies for the project and conclude with decisions about each.

2.4.1 Client Side Framework

Angular

'Angular is an application-design framework and development platform for creating efficient and sophisticated single-page apps.'[7] It was developed by Google to provide a complete framework for simple to complex single-page apps. Due to the size of this framework and the fact that it utilises Typescript makes for a steep learning curve, however this is counteracted by the large number of resources available from it being hugely popular and backed by a large company.

React

'React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called "components".'[8] React allows the developer to use JSX to combine HTML and JavaScript into one file, although this is not required to benefit from React. Unlike Angular, this is a library and not a full framework. This will reduce the amount of learning required to use it, but it is likely extra libraries will be required to provide all the functionality required. There is also a lot of resources available as React is maintained by Meta and is the most popular of the frameworks considered according the 2022 Stack Overflow Developer Survey[9].

Vue

'An approachable, performant and versatile framework for building web user interfaces.'[10] While not backed by a large company, Vue has grown in popularity and has many large commercial sponsors. This means there is likely to be less documentation available, but this library is relatively simple compared to the options considered above.

Conclusion

After reviewing the three frameworks, the decision was made to use Angular for the project. This is due to prior experience with the framework, and with no background in JavaScript the increase in learning curve from Typescript is negligible. However, any of these choices would have been suitable for the project and the decision is mostly down to personal preference.

2.4.2 Server Side Framework

Due to the previously mentioned lack of JavaScript experience, the decision was made to only consider Python based frameworks to reduce the number of languages to learn.

Django

'Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.'[11] As with Angular, this can be used as a full framework and so, while it will contain almost everything required, there will also be a lot of unnecessary features included. Django was designed to support rapid development, as shown by the slogan 'The web framework for perfectionists with deadlines.'[11]. It features database abstraction which allows developers to create 'Models' to represent the data stored in a database, handling communication with the database avoiding the need for coding in SQL.

Flask

'Flask is a lightweight WSGI web application framework.'[12] It is a microframework that started as a wrapper for Jinja and Werkzeug and has grown into a popular web framework. Flask does not provide much functionality that is already provided by another extension, this keeps Flask small but can introduce complex library requirements.

Conclusion

Django was chosen here, again due to prior experience, particularly in using Django and Angular together. While using two large frameworks may result in the project suffering from an amount of bloat, the prior experience means completing the project and having time to refine it is more likely.

2.4.3 Database

A graph database is a type of NoSQL database that uses nodes, edges, and properties to represent and store data. It is often described as storing data as it would be drawn on a whiteboard. This approach makes querying relationships in the data much faster as they are embedded in the data, rather than using JOIN operations or cross lookups often seen in SQL implementations. The underlying storage mechanism of a graph database can vary, some depending on an abstraction to store the data in a typical table based manner and some opting for a 'native' approach, maintaining the graph structure throughout the system.

Neo4j

'Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend for your applications'[**WhatGraphDatabaseb**] Neo4j provide various tools for developing with graph databases, only the Aura platform will be considered here as it is the direct comparison for the other tools discussed, and a managed service is ideal for this project. A free database instance is provided in AuraDB for each account which allows up to 200000 nodes and 400000 relationships. Aura features a data import tool that allows developers to create a model of the database structure which can then be populated with data from CSV files. It also has an explore tool, Bloom, this allows developers to view the database graphically and queries can be performed to show expected outputs. Neo4j provide a lot of useful documentation and tutorials for all the services they provide, this includes free e-books and course style content to guide users through the material.

Amazon Neptune

'Amazon Neptune is a purpose-built, high-performance graph database engine optimized for storing billions of relationships and querying the graph with milliseconds latency.'[1] Neptune is a managed graph database service provided by

AWS. It offers a high throughput and low latency system that automatically scales with demand. As to be expected from any AWS product it is also highly secure and fault-tolerant. However, this comes at a cost; a free trial is offered for 30 days and after that a monthly fee is incurred with additional costs for data transfer, backups, and storage consumption.

Conclusion

The decision was made to use Neo4j. This is primarily because the Aura platform provides a permanent free database instance which has ample resources for this project. There also exists a Python libraries, neomodel and django-neomodel, for easily integrating database access using Django Models.

2.5 Conclusion

Throughout this chapter research has been conducted that will continue to provide benefit throughout the design and implementation stages. The review of existing solutions has highlighted the need for this application and has provided insight into how existing solutions have addressed the issue. The technology review has made it clear which technologies should be taken forward in the project as well as providing some background on why these technologies are needed and how they work.

Chapter 3

Design

3.1 Introduction

3.2 System Design

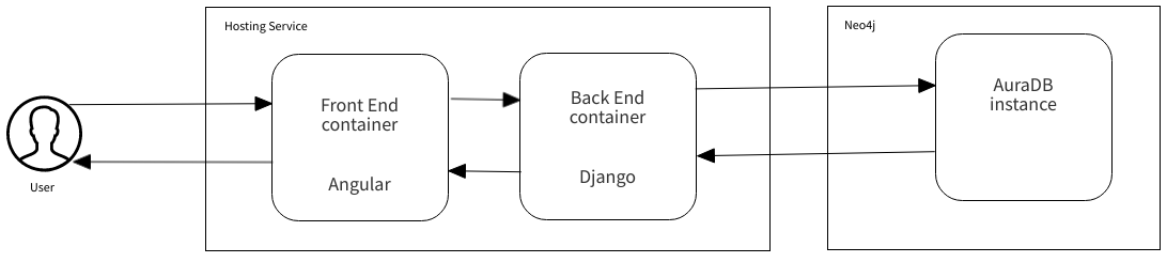


Figure 3.1: System Architecture Diagram

The system architecture for this project is similar to most web based applications as the only real difference is the type of database used. This means the user interacts with the web page created by the front end server, this then communicates with a separate back end server using the defined API. If needed the backend server can then communicate with the database, which in this case is hosted remotely by Neo4j. The front and back end servers would likely be hosted in the cloud via the use of some containerisation system such as Docker.

3.3 UI Design

Given that the data is stored in a graph database and that is the focus of this project, it seemed obvious to display the data to the user as a graph. This idea was further reinforced by looking at the Neo4j tool Bloom. As shown in

the screenshots below, Bloom is used to display the data in a dynamic and responsive graph. The main view area is called the "Scene" and just this alone allows the user to quickly visualise the data and see the relationships formed. In the scene the nodes can be dragged using the mouse and the graph updates using a physics simulation to move the other nodes. This helps ensure the other areas of the graph are still readable while still allowing the user to manipulate the graph.

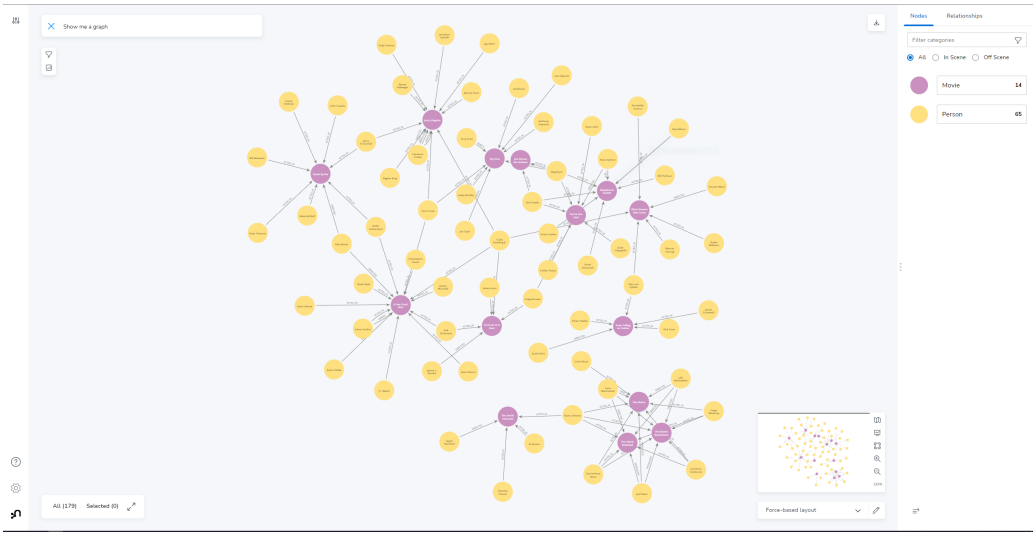
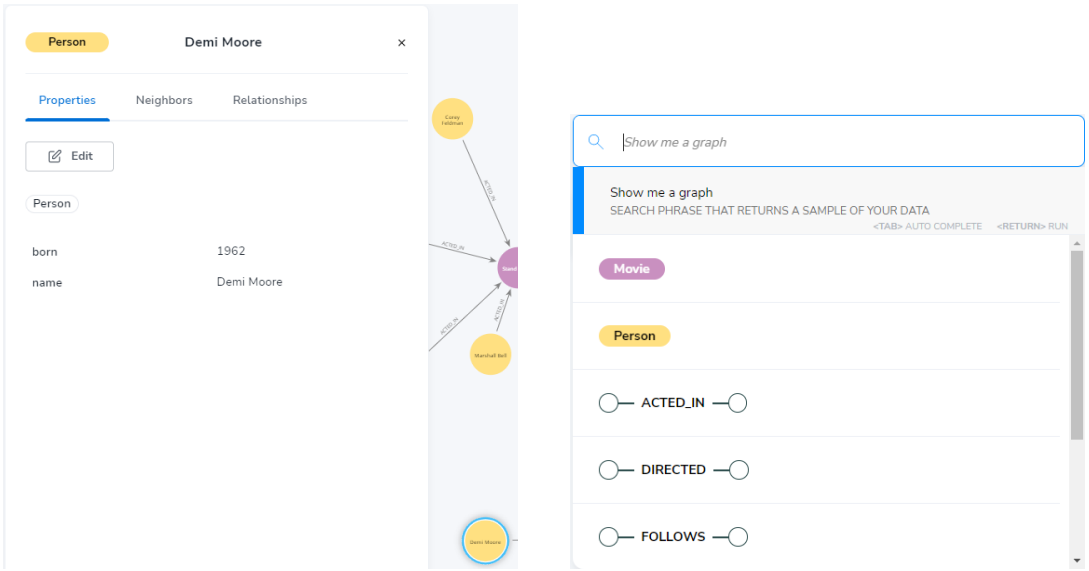


Figure 3.2: Neo4j Bloom usig the demo Movies database

The following screenshots show two key features of Bloom. The first is the inspect element which allows the user to view the full properties of a node or relationship by double-clicking them. This means the graph itself is not cluttered with extra information but the user can still easily access that data when required. The other feature is the search bar which has a few unique functions that utilise the graph structure of the database. It uses the types of nodes and relationships in the database to provide some quick pre-filled searches, For example Actor - ACTED_IN - Movie with the demo database. This can be used to quickly filter out unwanted data from the scene without having to know how to write custom query statements. The second functionality follows on from this, in that the user can define their own query statements which become a command available in the search dropdown. This is useful if the user is familiar with Neo4j's own query language CYPHER, and they want to leverage that for more powerful querying.



(a) Bloom Inspect Element (b) Bloom Search Element

Figure 3.3: Neo4j Bloom inspect and search elements

These are all features that should be considered when implementing the user interface, however not all of them are essential for a usable application.

3.4 Conclusion

Chapter 4

Implementation

4.1 Introduction

4.2 Tools

4.2.1 IDE

Visual Studio Code was used for this project due to prior experience with the IDE. It is also one of the most popular tools in the industry as shown by the Stack Overflow Developer Survey[9] and the TOP IDE Index[13]. VS Code has excellent support for many programming languages, and with a wealth of community made extensions there are many tools to aid with development.

4.2.2 Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.[14] Using version control with an external hosting provider also makes working over several computers easy which will be useful for this project, and it ensures everything is backed up remotely. Git was chosen for version control as it is the industry standard, with GitHub being used as the hosting provider.

4.2.3 Database Visualisation

Neo4j has two tools for database visualisation as part of the AuraDB web interface; Bloom and Browser. Bloom is used to visualise the data in a graph as has been discussed in the design chapter previously. Browser is used to test CYPHER queries on the database. CYPHER is the query language created by Neo4j for retrieving data from their graph databases. As shown in the screenshot below, the user can enter a query and have the data returned as a graph,

table (represented as a series of JSON objects), raw text and as code (JSON objects). This is useful for quickly testing CYPHER queries and debugging database interactions.

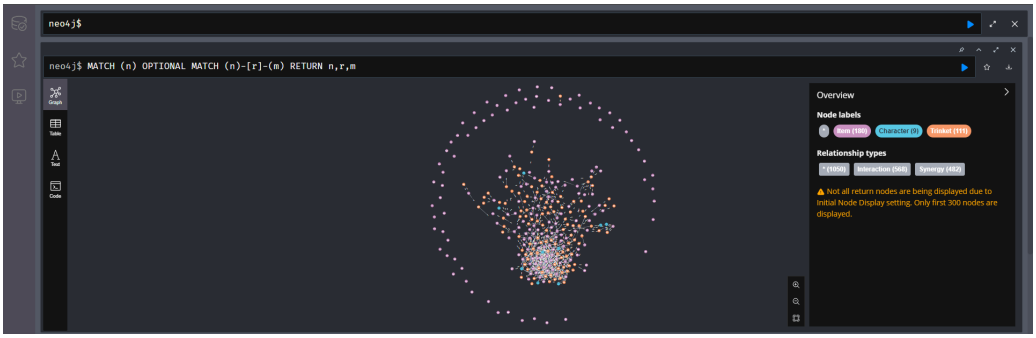


Figure 4.1: AuraDB Browser

4.2.4 Libraries

4.2.5 Client Side

cytoscape - used to create graphs on the front end rxjs - observable thing material - googles own library of components to make ui quickly

4.2.6 Server Side

django-cors-headers - allows in browser requests to djano application from other origins djangorestframework - ??? neomodel - used to create models for getting data from neo4j database pylint - used to enforce PEP8 coding standards for code quality black - autoformats code to ensure readability

4.3 Data Processing

this section covers getting the data needed to create the database and creating the database

4.3.1 Finding Data Source

fandom wikis all should have an xml dump available on the special:statistics page It isnt always up to date, so i had to request a update while waiting for that I looked at other BOI sources such as platinum god and the game files

got the xml file and its massive and not consistently formatted it's set up like a huge list of pages but theres not order to them and sometimes random stuff is added in between them looked challenging to extract data from so looked

towards other games i.e. pokemon however that wiki didn't have a dump at all and i got no response regarding getting one other data sources for pokemon exist but each have their own issues

4.3.2 Data Extraction

while looking for alternative sources and waiting on response from wiki admins worked on extracting the data used beautifulsoup to parse the xml tags and traverse the data Used the items collection page to get the names of all the items locate the correct xml elements trinkets and characters(?) had to be hardcoded as a complete list didnt exist in the xml

beautifulsoup is used to traverse the data and get the 'text' child of all the relevant 'pages' then regex is used to grab the relevant sections from the text

4.3.3 Cleaning the Data

regex is used again to clean the sections of text The text contains lots of tags for vairous features in the wiki such as dlc tags and links to other pages turns bullet pointed lists into nested arrays

4.3.4 Importing into Database

neo4j has a data importer tool which uses csv files to populate a data model csv files were generated from the data objects in python

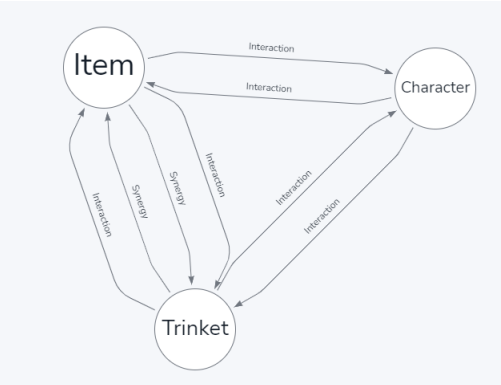


Figure 4.2: Database Model in Neo4j Data Importer

4.4 Web Stack

creating django project with an api app and a main "query" app api app contains webserver (?) and the settings each app then contains the the models,

views and urls etc urls.py defines what functions get called by each urls views.py defines the functions used by urls, these functions contain all the functionality the user interacts with, each function takes in a request from the url and depending on the type of request (e.g get, post etc), performs the relevant action models.py contains classes each of which represent a type if data entity

create a angular project and any components (explain angular project structure) each component represents a "screen" (explain component structure)

There is a shared services file, this defines the interaction with the backend api the address of the backend server is defined in there and the functions that make the http requests

each project has it's own git repo which are then submoduled into the main project repo

4.5 Database Interaction

Used neomodel to define classes for each entity and relationship type, these classes contain methods for retrieving data Getting all nodes is very easy due to built in methods, however these do not exist for relationships so to get all relationships you have to perform a cypher query to get them

4.6 Client Side

first made tables to check everything worked found that the json like object created by neomodel couldn't be interpreted by the front, so had to manually make a get method for each class This causes getting large amounts of data (relationships) to take quite a long time

I created a second component to show the data in a graph form as per my initial design To do this I used cytoscape.js (more info) had to rewrite how the data is formatted when sent to the front end so that it matchs what cytoscape expects still takes a long time to get data from backend so decided to write it to a json file so at least for quicker debugging it can load directly from the file tweaked some settings to make the graph easier to read

4.7 Conclusion

Chapter 5

Evaluation

5.1 Introduction

5.2 Project Evaluation

5.3 Future Work

5.4 Lessons Learned

5.5 Conclusion

Appendix A

Appendix

References

[1] *What Is a Graph Database?* Amazon Web Services, Inc. URL: https://aws.amazon.com/nosql/graph/ (visited on 01/06/2023).

[2] *The Binding of Isaac: Rebirth on Steam.* URL: https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/ (visited on 01/31/2023).

[3] *The Binding of Isaac Wiki.* URL: https://bindingofisaac.fandom.com/wiki/The_Binding_of_Isaac_Wiki (visited on 01/09/2023).

[4] *Binding of Isaac: Rebirth Wiki.* URL: https://bindingofisaacrebirth.fandom.com/wiki/Binding_of_Isaac:_Rebirth_Wiki (visited on 01/09/2023).

[5] *Isaac Cheat Sheet - Platinum God.* URL: https://platinumgod.co.uk/ (visited on 01/09/2023).

[6] *Frequently Asked Questions - Isaac Cheat Sheet - Platinum God.* URL: https://platinumgod.co.uk/faq (visited on 01/09/2023).

[7] *Angular - Introduction to the Angular Docs.* URL: https://angular.io/docs (visited on 01/09/2023).

[8] *Tutorial: Intro to React – React.* URL: https://reactjs.org/tutorial/tutorial.html (visited on 01/09/2023).

[9] *Stack Overflow Developer Survey 2022.* Stack Overflow. URL: https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022 (visited on 01/10/2023).

[10] *Vue.Js - The Progressive JavaScript Framework — Vue.Js.* URL: https://vuejs.org/ (visited on 01/09/2023).

[11] *Django.* Django Project. URL: https://www.djangoproject.com/ (visited on 01/09/2023).

[12] Armin Ronacher. *Flask: A Simple Framework for Building Complex Web Applications.* Version 2.2.2. URL: https://palletsprojects.com/p/flask (visited on 01/09/2023).

[13] *TOP IDE Top Integrated Development Environment Index.* URL: https://pypl.github.io/IDE.html (visited on 04/23/2023).

[14] *Git - About Version Control.* URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (visited on 04/23/2023).