May 2022

SCHOOL OF COMPUTING AND MATHEMATICAL SCIENCES, UNIVERSITY OF LEICESter

CO3201 Computer Science Project: Dissertation

LEGO: SET CHECKLIST CREATOR

DAN Booth

## contents

## Abstract

## Introduction

Ever since I was young, I have always loved building and playing with Lego, getting Lego sets regularly for Christmas and birthdays. Over time as I got more Lego sets, I had to take some older sets apart to make room for newer ones. The Lego pieces from these sets would be stored in separate containers so that if I wanted to rebuild a set, I could simply get that certain container. However, as I got more Lego sets these pieces became muddled so that it was no longer as simple as picking a container. This can be done using the list of pieces in the back of the Lego set’s instruction booklet and as I find the pieces tick them off this page. This can work when rebuilding the set for the first few times but after a while, it can become very difficult because there are ticks all over the page. Therefore, having a digital checklist for pieces in a Lego Set would help fix this issue as once you have built the set, the next time you go to rebuild it the checklist will be blank, and you can start all over again.

For example, you have a Lego set that you have taken apart and put all the pieces in a box along with other Lego pieces, and you would like to rebuild the set, you could do this easily using a digital checklist.

The target users for my project will be 18+ Lego builders, enthusiasts, and collectors (referred to as ‘Lego enthusiasts’ from now on for briefness) who are looking to build or rebuild a Lego set they own.

### Aim

The key aim of this project is to create a digital checklist for pieces in a Lego Set. Users can find and then select a Lego Set they would like to see the pieces for. They can then view all the pieces in this Lego set (like in the back of the instruction book), and they can check they have all the pieces when they are building the set again.

### Objectives

1. Write a program to connect to a Lego Set API and retrieve data
2. Learn how to turn JSON files into a class
3. Build an application using the Spring Model-View-Controller (MVC) framework
4. Design a cross-platform website (view) from which users can use the system
5. Implement a cross-platform website (view) from which users can use the system
6. Design a way for users to save progress on a checklist
7. Implement a way for users to save progress on a checklist

## Survey of Literature/Information Sources

### Preliminary Research

To begin with, I looked up the Rebrickable API [1] that contains data for Lego sets, that I am going to use, that can search through to find a Lego set and the pieces within this set. I also read the documentation for the API [2], finding out that it's a RESTful API (meaning I can use HTTP requests to access data) and to access the data I need an API key that is freely available with an account. Using the API you can request a Lego set directly using the Lego set unique number, or search using “A search term”, filter using “theme\_id (a number associated with a Lego theme e.g. Star Wars, that can be retrieved also using API), min\_year, max\_year, min\_parts, max\_parts” and order by a certain “field” (“set\_num”, “name”, “year”, “theme\_id”, “num\_parts”). Data is returned from the API in the form of JSON files, and a set returns “set\_num”, “name”, “year”, “theme\_id”, “num\_parts” and “set\_img\_url”, but to retrieve a JSON of all the pieces in a Lego set another call to the API needs to be performed. This returns a list containing each part however this cannot be ordered using the API.

### Questionnaire Introduction

I then performed some data collection on what my target users would like from a digital checklist for pieces in a Lego Set, via an online questionnaire (see **Appendix A**). Using this I could identify their key requirements and features for the system, for example where they would like to use the system, how they currently check they have all the pieces for a Lego set, other tools they use for research, how important certain features would be to them and if they have any other ideas for features.

### Questionnaire Results and Further Research

My questionnaire received a reasonable number of responses (20 responses) and from the results of this questionnaire, I was able to decide on certain features and requirements for my project. (For full results to questionnaire see **Appendix B**).

Chart, pie chart

Description automatically generated

Figure 1 – Question 1 Results

Question 1 showed me that the majority of users (75% see Figure 1 above or **Appendix B**) would like to use the digital checklist on both PC/Laptop and mobile devices, which helped inform me that my digital checklist for pieces in a Lego Set should run on both these types of systems.

I also learnt from questions answers to 2 and 3 (see **Appendix B**) that Lego enthusiasts who use a digital tool use the website Bricklink [3]. I found that on Bricklink users can add pieces from a Lego set to a “wanted list” and from there tick of parts you have. This shows the user how many pieces they need and how many they currently have found. However, this number easily be changed by accident which could cause issues. For example, users could believe they have all the pieces for a set but they accidentally decreased how many pieces they needed so are missing one, or the opposite where they increase the number they need but have all of them. Users can’t filter pieces by colour or type making it difficult to find pieces, also when pieces are fully found they are not hidden from the list. Any pieces missing can easily show a list of possible locations to buy them. Most of these issues appear because the purpose of the tool is to buy pieces for a Lego set.

The answers from question 3 (see **Appendix B**) also show some people currently use the Rebrickable website [4]. On Rebrickable, which also provide the API I am going to use, users can find a Lego set by typing in the set number or searching by a text search (i.e. Set Name) and filter by a range of year released, range of the number of parts and also filter by themes. On the page of a set (e.g. this Lego Set [5]) users can see a list of all parts, the instructions, pictures of the Lego set, year released, number of parts etc. Here if the user has an account they can add the set parts to a List. On the list, the user added parts too, users can filter by piece colour, type (Category) and sort by colour, Hue, part, type (category) and price to buy the Lego piece. Users can see how many each piece is required as well as the colour and price to buy it but to check a piece off the list, the user has to delete it from the list meaning you can’t undo the change, also users can change the number of certain pieces needed but not see the original number (like BrickLink see above). This is primarily due to the fact the tool is meant to help users buy Lego pieces for a set, also the same as Bricklink, but can be used as a makeshift checklist.

Overall, the results of questions 2 and 3 (see **Appendix B**) has helped me find and research similar software and helped give me ideas on what would be useful to use from them.

Chart, bar chart

Description automatically generated

Figure 2 – Question 4 Results

The results of question 4 (see Figure 2 above or **Appendix B**) provided lots of useful information about how users would like to search for a Lego set. Some of the answers were very conclusion for example 19 people said that searching by ‘Age Range’ for a Lego Set was not need showing me that there is no demand for this search parameter. Likewise, all 20 respondents stated they would like to be able to search by ‘Set Number’.

The answers to the ‘Set Pieces’ section of question 4 (see **Appendix B**) are quite varied, with 8 responses saying it was ‘not needed’, which was one more the ‘Filter by’ (7 responses) and one less than ‘Sort by’ (9 responses). This shows filter and sort for ‘Set Pieces’ only just make up the majority of responses showing that maybe this is not a key requirement when searching for a set but would still be useful.

Overall conclusions that I can draw from question 4 is that overall users would like to search by ‘set number’ and ‘set name’ when trying to locate a Lego Set. They would also like to filter by the ‘year made’, ‘theme’ and ‘Set Pieces’ as well as sorting by ‘Theme’, ‘Year Made’ and ‘Set Pieces’.

The results of Question 5 (see **Appendix B**) show that most people find it important or very important that a digital checklist the Lego pieces can be sorted by colour and type of piece, showing this should be a key feature of my digital checklist. However, being able to filter by Lego pieces type and colour are shown to be not as important and therefore are not as key to people. Finally, having a link to buy a missing Lego piece and being able to scan Lego pieces in a set to see if they are there and then check them off, have very mixed answers showing they should be nothing more than optional for the checklist.

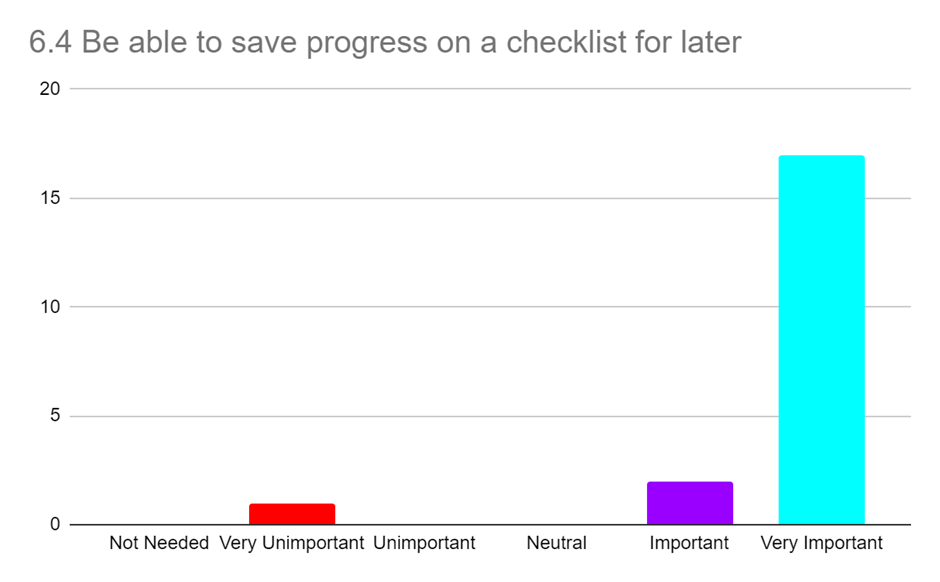


Figure 3 – Question 6 part 4 (Be able to save progress on a checklist for later) Results

Question 6 results (see **Appendix B**) shows that it is very important to most people (17/20 see Figure 3 above) to be able to save progress on a checklist making this a key requirement for the system. Being to view and download instructions is also important to most users, as is being able to save Lego sets they own to a list meaning this is also key. The responses to having a favourites list for Lego sets are very mixed but mostly positive showing that it would be nice to have but not key.

From the responses to question 6 I went and found the Brickset API [6] that requires a free API key, and I can use it to retrieve Lego set instructions (as the current Rebrickable API cannot do this). These are returned as a list of instruction PDF links, in a JSON file. This API also allows users to search for Lego Sets, but I will only use this API for retrieving instructions as it does not contain data on pieces within a Lego set, which is a vital part of the project.

Graphical user interface, text, application, email

Description automatically generated

Figure 4 – Question 7 Results

Finally, the results of question 7 (see Figure 4 above or **Appendix B**) where users are allowed to add any ideas for any other features gave some useful ideas. For example, being able to also scan bricks with a webcam as well as a mobile for PC/Laptop users. As well as if there are multiple Lego pieces of the same type and colour on the checklist being able to record the specific number of these found. Finally, another feature suggested was to import and export XML files like a Bricklink [3] wanted list. I took these suggestions into account when designing my requirements.

## Design

### Requirements

#### Key Features:

* The system must be usable as a website on both mobile and PC/laptops
* The system must display a list of all Lego sets stored in Rebrickable API [1]
* The system must have a search feature that allows users to search a list of Lego sets. Can search by set number and text search (e.g. set name), and filter and sort by year made and set theme.
* Users must be able to ‘check’ piece off the checklist, showing how many more of that piece are remaining
* The system must show on the checklist (for Lego pieces in a set) a picture of the piece, with correct colour, as well as an alternative text description including piece name and colour
* Users must be able to sort a checklist by colour and type of a Lego piece
* The system must be usable with and without a user account
* Users must be able to save progress on a checklist

#### Nice to have Features:

* The system may have an additional search parameter to sort and filter by number of pieces in a Lego set
* The system may have an additional search parameter to sort alphabetically by Set Name
* The system may have a consistent and simple UI
* Users may be able to view instructions for a Lego set
* Users may be able to download instructions for a Lego set
* Users may be able to filter a checklist by the colour of a piece
* Users may be able to filter a checklist by type of a piece
* The system may have a link to buy a missing piece from a Lego Set
* Users may be able to create an account
* Users with an account may be able to save sets they own to a ‘Sets Owned List’, so they can easily find them later

#### Optional Features:

* Users with an account could create lists for Lego sets and save sets to them, so they can easily find them later (Sets can be in multiple lists)
* Users could search their ‘Sets Owned List’ and other lists for Lego sets, like the main search feature
* The system could save users progress on a checklist to the database
* The system could also be a mobile application
* Users could scan Lego pieces with a phone camera or webcam to check if the piece is in a Lego list
  + If it is in the set (and not already enough of them), there is an option to check pieces off the Digital Checklist
  + If in the set but already have all that type of piece needed, it will inform the user of this
  + If not in set it will inform the user of this
* Users could import and export a checklist as an XML file in the Bricklink [3] wanted list format

### High-level overview of the architecture of the system

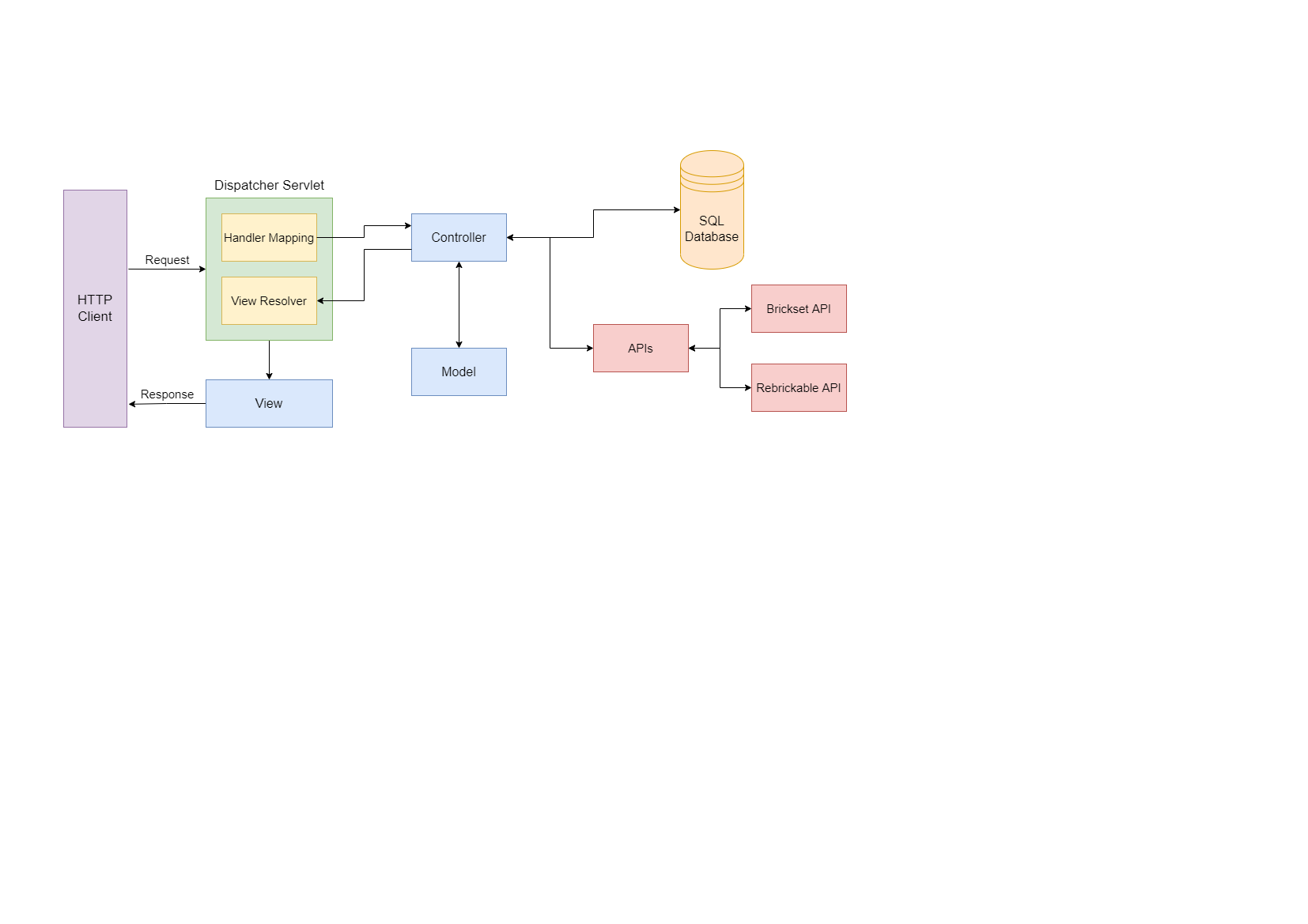


Figure 5 – High-Level Overview

Figure 5 above shows a high-level view of the Spring MVC architecture that my website will use. Where the view will be the JSP that is displayed to the users on the HTTP client (their web browser). When the user interacts with the View, via the web browser, a request is sent to the Dispatcher Servlet. Here the Dispatcher Servlet will use the Handler Mapping to match the request URL to the correct Controller. The controller will then call APIs or interact with the SQL database to collect and edit information as needed, it will then update attributes in the Model, before returning the name of the next View to the View Resolver. View Resolver, which locates the correct View add adds in the Model attributes. This View is then sent back to the HTTP client as a response.

### APIs

#### Rebrickable API

The Rebrickable API [1] (as mentioned above see 3 Survey of Literature/Information Sources) stores all the data about all Lego sets and the Lego pieces in these sets, as well as all the Lego themes and which sets are in them. This API will be used to search for Lego sets and retrieve all the Lego pieces in a Lego Set.

#### Brickset API

The Brickset API [6] (as mentioned above, see 3 Survey of Literature/Information Sources) will be used to obtain PDF instructions for a Lego set so users can view and download these for a Lego set.

### Database Design

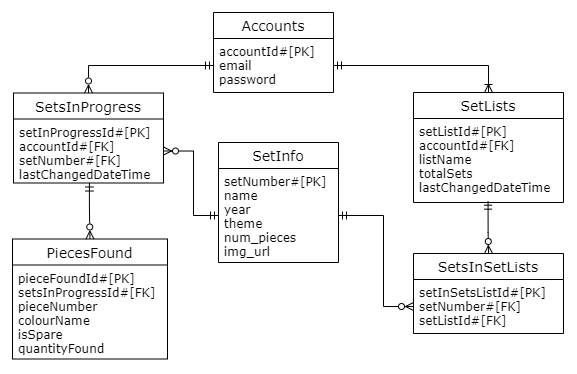


Figure 6 – ER Class Diagram

I have used SQL for my database, and Figure 6 above shows an ER class diagram for this database, that may change depending on the requirements completed. The database has several tables called Accounts, SetsInProgress, PiecesFound, SetLists, SetsInSetsList and SetInfo. These database tables are generated using hibernate [7] and JSP in classes described in the 4.5 Class Diagram.

The Accounts table is used to store user accounts, with ‘accountId’ as the primary key, ‘email’ which is a unique attribute as an email can only belong to one account, and a ‘password’ that is encrypted using a hash and salt.

The SetsInProgress table stores the set numbers of sets (‘setNumber’) that a user currently has a checklist in progress for, the information for these sets is stored in the ‘SetInfo’ table explained below. ‘setInProgressId’ is the primary key, with the user’s ‘accountId’ as the foreign key so it's easy to find which user it belongs to. Finally, ‘lastChangedDateTime’ which stores the last time the set in progress was saved to the database, this is used so that the last 3 used sets in progress can be displayed to the user on the home page.

The PiecesFound table is used to store pieces for a ‘SetInProgress’ and contains attributes ‘pieceFoundId’ is the primary key, ‘setInProgressId’ is a foreign key so it's easy to identify which setInProgress the pieceFound belongs to. The ‘pieceNumber’, the ‘colourName’ is the colour of the piece, ‘isSpare’ shows if this is a spare provided piece with the Lego Set but not required to build the set and ‘quantityFound’ is the quantity of the particular piece that has currently been found (if 0 pieces have been found these won’t be saved to save database storage).

SetLists table stores lists the user has created, that contain Lego sets (called setlists). The table stores the name of the list (‘listName’), a primary key (‘setListId’) (used to find the sets that belong to the list in the SetsInSetsList table explained below), the total number of sets in the list (‘totalSets’), with the user’s ‘accountId’ as the foreign key so it's easy to find which user it belongs to. Finally, ‘lastChangedDateTime’ which stores the last time the setlist was saved to the database, is used so that the last 3 used setlists can be displayed to the user on the home page.

SetsInSetsList stores the set numbers of sets (‘setNumber’) saved to a setlist in the database, the information for these sets is stored in the ‘SetInfo’ table explained below. It has a primary key ‘setInSetListId’, ‘setListId’ is a foreign key so it's easy to identify which setlist this belongs to and the ‘setNumber’ for the set, which can only appear once in a setlist.

The SetInfo table holds information about sets stored in either the SetsInProgress or SetsInSetLists tables. ‘setNumber’ is the primary key that is used to link to this table in the other tables, the set’s ‘name’, ‘year’ released, ‘theme’, number of pieces in the set ‘num\_pieces’ and a URL for an image of the set (‘img\_url’).

### Class Diagram

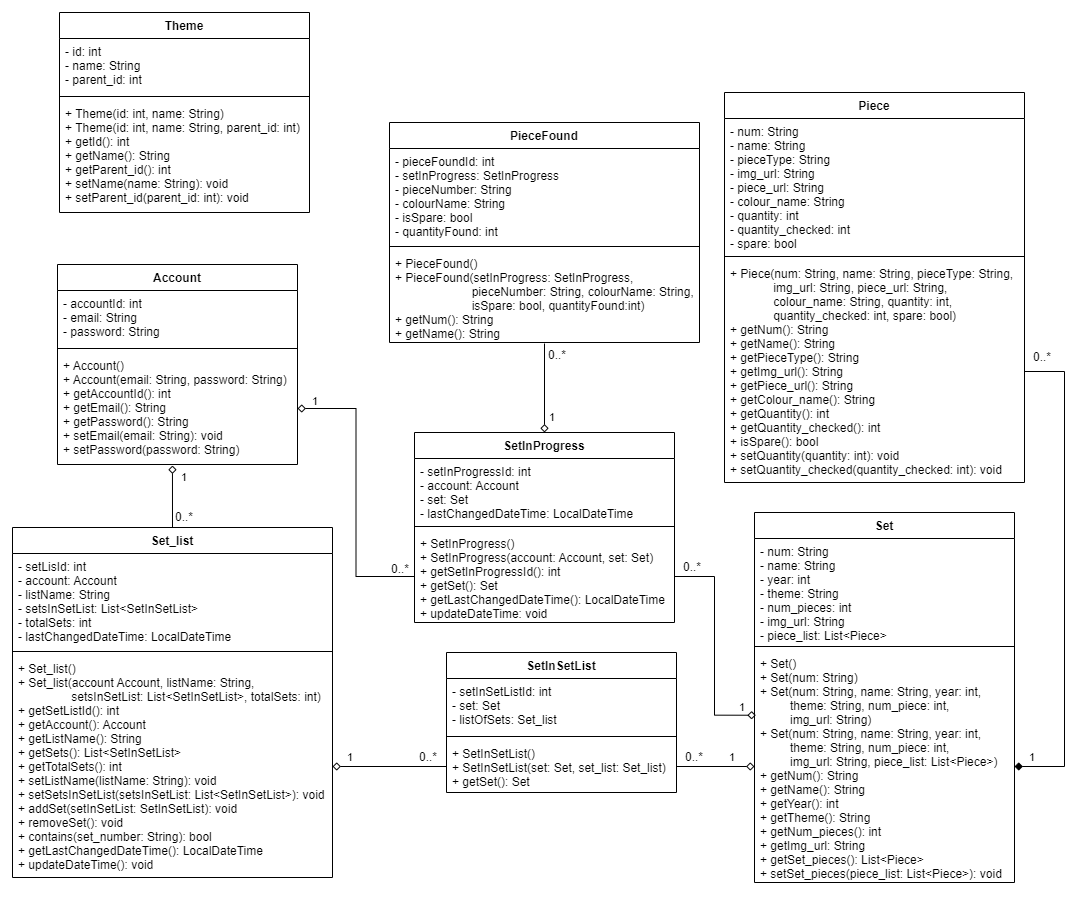


Figure 7 – Class Diagram

Figure 7 above is a class diagram for my system, with several classes Set, Piece, SetInSetList, Set\_list, SetInProgress, PieceFound, Account and Theme.

The Set class is used to store Lego set data from a JSON file received from the Rebrickable API [1]. This data includes the Lego Set’s Number (‘num’) which is a unique number that identifies the Lego Set, and this is a string because all set numbers contain a '-' that is followed by a version number that donates different versions of certain sets, with the standard being '-1'. It also contains the ‘name’ of the Set, the ‘year’ released, the ‘theme’ the set belongs to, the number of Lego pieces (‘num\_pieces’) in the set, a URL for an image of the set (‘img\_url’). Finally, there is an attribute called ‘piece\_list’ which is a list of the Piece class and is used to store a list of all Lego pieces in a Lego set.

The Piece class is used to store information about a Lego piece from a JSON file received from the Rebrickable API [1]. This contains a piece number (‘num’) as these sometimes contain letters this is a string, ‘name’ of the piece, a ‘pieceType’ which is the type of Lego piece the piece is, a URL for an image of the piece (‘img\_url’) and another URL (‘piece\_url’) to the piece’s page on the Rebrickable website where the user can buy the Lego Piece [6]. A ‘colour\_name’ that’s the colour of the piece, ‘quantity’ which is the number of these pieces that are required to build the Lego Set, ‘quantity\_checked’ which is the quantity the user has checked off this pieces and finally ‘spare’ which denotes if this piece is a spare provided with the Lego Set but not required to build the set.

The SetInSetList class is used to link the Set\_list and Set classes and is used for the database (using JPA annotations [7]) so that information on a Lego set does not have to be duplicated if in several setlists, and can just save a table that links to the set via the set’s number. It contains a unique auto-generated id (‘setInSetListId’) used as the primary key for the table in the database, ‘listOfSets’ is the setlist the set is in and ‘set’ is the set that is in the setlist.

The Set\_list class is used to store a list of Lego sets retrieved from a user’s saved list of Lego sets from the database, this class is also used to create the database using JPA annotations [7]. It contains the name of a list (‘listName’), the user ‘account’ the list belongs to, ‘setListId’ is a unique auto-generated id used as the primary key in the database, the number of sets in the list (‘totalSets’), ‘lastChangedDateTime’ which holds the last date and time the class was changed and is used so that the last 3 used setlists can be displayed to the user on the home page and so that the last accessed list is at the top of the selector when selecting a setlist to add a Lego set to. Finally, there is a List of type SetInSetList (‘setsInSetList’), that is used to store all the Lego sets in the list. It contains several functions with the ‘contains()’ that checks if a set is in the list, ‘updateDateTime()’ sets the ‘lastChangedDateTime()’ to the current date and time, ‘addSet()’ adds a set to the list and increases total sets by 1, and ‘removeSet()’ that decreases total sets by 1.

The SetInProgress class also uses JPA annotations [7] to generate a table that connects a user’s account to a set checklist that they have saved to the database. Similarly, the SetInSetList class links to the Set class so that in the database set information is not saved multiple times, by having a variable ‘set’ of type Set. This also contains the user ‘account’ the set in progress belongs to, ‘lastChangedDateTime’ which holds the last date and time the class was changed (used like Set\_list to display the last 3 last saved sets in progress), and ‘setInProgressId’ is a unique auto-generated id used as the primary key in the database. This also contains a ‘updateDateTime()’ which operates in the same way as in the Set\_list class.

The PieceFound class is another class that is used to create the database (using JPA annotations [7]). This class is used to store pieces from SetsInProgress, and only contains information that makes each piece unique, the ‘pieceNumber’ which is unique, the piece’s colour (‘colourName’) and if it is a spare piece (‘isSpare’) and finally how many of this particular Lego piece have been found (‘quantityFound’). It also has an auto-generated primary key ‘pieceFoundId’.

The Account class is used to create the database table (with JPA annotations [7]) that holds an account user’s information. It has a user’s auto-generated primary key (‘accountId’), the users ‘email’ and ‘password’ that are used by a user when logging into the website.

Finally, the Theme class holds information about Lego Set themes retrieved from the Rebrickable API [1] as a JSON file. This class is separate from the other tables and is used to store all the themes, so that each time a set is received from the Rebrickable API [1], the program does not have to call the API again with a theme number to retrieve that theme’s name. It contains a unique ‘id’ used to identify the theme, the theme’s ‘name’ and ‘parent\_id’ which is the id of the theme that this theme is a sub-theme for.

### User Interface Design

#### Home Page

Graphical user interface, text, application

Description automatically generated

Figure 8: Home Page - logged out

The first page that appears to the user when they access the site is the home page (Figure 8), here they can see a description of the website and what it is used for, helping useability by making it clear what the website does. There is also some text informing the user that they are not logged in, offering users the option to login or create an account, but also making it clear to users that they can continue without these.

At the top of the page is a black navbar containing the websites name so it’s clear and obvious to users what website they are on. Then there are 2 number boxes where users can enter a Lego set number and variant number (at least 1) and a button that will search for the Lego Set and return Figure 12 or Figure 13, these are explained in 4.6.3 Set Page below. Finally, there is a ‘Sign in’ navbar link which opens the Figure 18: Login Popup explained in 4.6.7 Login and Sign-Up Popup below.

Underneath the navbar is a secondary navbar with a ‘Search’ link that opens the Search page (4.6.2 Search Page) and a ‘Import Checklist’ button that opens Figure 17: Import Checklist Popup.

The breadcrumb trail, in the footer at the bottom of the page, has the current page in white text, informing them of their current position in the user journey.

Graphical user interface

Description automatically generated

Figure 9: Home Page - logged in

The home page displayed to a logged in user (Figure 9) is mostly the same but with a few differences. The ‘Sign in’ navbar link has been replaced with ‘Logout’, which when clicked opens Figure 20: Logout Popup, explained later in 4.6.8 Logout Popup.

Additional links are now shown to the user on the secondary navbar, with links to view their ‘Set Lists’ (see 4.6.10 Set Lists Page), ‘Sets In Progress’ (see 4.6.9 Sets in Progress Page) and their account ‘Profile’ page (see 4.6.16 Profile Page).

The not logged in message is replaced with the users three last edited set lists, with links that can be used for quick access to view these (see 4.6.13 Set List Page below), along with the users three last saved sets in progress (see 4.6.9 below).

#### Search Page

Table

Description automatically generated

Figure 10: Search Page - Filter Bar Open

Table

Description automatically generated

Figure 11: Search Page - Sort Bar Open

The Search Page (Figure 10 and Figure 11) can be accessed clicking the ‘Search’ link on the secondary navbar on the Home Page (4.6.1), or the ‘BACK TO SEARCH’ button on the Set Page (4.6.3) or breadcrumb links on other pages. On this page users can search for a Lego Set they would like to view.

In the centre of the page a list of Lego sets is displayed in rows, with columns displaying different bits of a Lego sets information. The blue square represents an image of a Lego Set that when clicked will open Figure 16 to view the image enlarged (see more detail below). There is a ‘plus’ icon button that only appears to signed in users, that when clicked opens a popup (Figure 26) to add a set to a setlist.

Above the list of sets is a header that has the name for each column, so it is clear to the user what each of these columns contains. Each of these column names (excluding ‘Set Image’), has an arrow next to it, these show if the sets are being sort by that column, with a down arrow showing the list is being sorted ascending in that column, an up arrow shows descending and an up-down arrow shows the sets are not being sorted by that column. For example, in Figure 10 the sets are being sorted by ‘Set Number’ ascending and not by any other column.

Clicking one of these arrows will sort the sets in the column the arrow is adjacent to. If a down arrow is clicked the sets will be sorted descending and the arrow will change to an up arrow, if it’s an up arrow they will be sorted ascendingly, and the arrow will change to a down arrow. If an up-down arrow is clicked the sets will be sorted ascendingly, and this arrow will change to a down arrow and any other arrows that are not up-down arrows will change to up-down as those columns are no longer being filtered. Clicking on an arrow to sort a list will remove any other sorts currently active including a multi-sort selected via the sort bar, which is explained below.

There is a scroll bar down the side of the page so users can scroll through this list of Lego Sets, and in the footer at the bottom of the page there are two buttons ‘Previous’ and ‘Next’, which go to the previous page of sets or the next page of sets, these are disabled if it is the first list of sets or last list of sets respectively.

If the user clicks a Sets set number, which is a hyper link, Figure 12: Lego Set Information Page will open displaying the selected set.

At the top of the page is a black navbar containing the websites name so it’s clear and obvious to users what website they are on, that when clicked acts as a link returning the user to the Home Page (4.6.1). Then there are 2 number boxes where users can enter a Lego set number and variant number (at least 1) and a button that will search for the Lego Set and return Figure 12 or Figure 13, these are explained in 4.6.3 Set Page below. Finally, there is a ‘Sign in/Logout’ navbar link which will appear as ‘Sign in’ when the user is not logged into an account, and ‘Logout’ when they are. ‘Sign in’ opens the Figure 18: Login Popup and ‘Logout’ opens Figure 20: Logout Popup, these are both explained later (in sections 4.6.7 and 4.6.8 respectively).

There is a navbar link ‘Filter’ which opens a filter bar underneath the navbar (shown open in Figure 10), closing the sort bar (shown open in Figure 11) if it is open. Likewise, the ‘Sort’ navbar link opens a sort bar underneath the navbar shown open in Figure 11, closing the filter bar if it is open.

The filter bar (Figure 10) contains filters the user can use to filter the Lego Sets, there is a text box to search by text (e.g. ‘Clone’), a select box to filter by Lego Set Theme (e.g. Star Wars, Lego City, Batman), ‘Minimum Year’ and ‘Maximum Year’ number boxes that filter between these years, ‘Minimum Pieces’ and ‘Maximum Pieces’ number boxes that filters the sets by the minimum and maximum number of Lego Pieces in the set and the ‘Filter’ button then applies these filters to all Lego Sets (via a call to the Rebrickable API [1]).

The sort bar (Figure 11) contains three select boxes so a user can perform a multi-sort, containing all the columns sorts ascending and descending with also the option of none. The first select must always be selected, and by default the sets are sorted by ‘Set Number’. This is used selecting a column to sort by and then the preceding column to sort by, up to a maximum of three columns, this means where the first column values of sets match then they are compared by the second column value instead e.t.c. There is then a ‘Sort’ button that when pressed applies this sort to the list. For example, in Figure 11 the sets are first being filtered by ‘Year Released’ ascending and then by ‘Set Number’ ascending, so the ‘Set 53’ and ‘Set 54’ both have ‘Year Released’ of 2001 so they are then compared by ‘Set Number’ and as ‘Set 53’ has a set number of ’53-1’ and ‘Set 54’ has a set number of ’54-1’ ‘Set 53’ appears before ‘Set 54’.

The breadcrumb trail, also in the footer, has the current page in white text, with the previous pages to the right with hyperlinks to those pages. I added this so a user can see their position on the user journey.

#### Set Page

Graphical user interface, application

Description automatically generated

Figure 12: Lego Set Information Page

Graphical user interface, text, application

Description automatically generated

Figure 13: Lego Set Not Found Page

If the Set Number entered is not found Figure 13 is displayed, this is mostly the same as Figure 12 (Lego Set Information Page) but the main body is replaced with a message informing the user that the set number entered was not found and with the same ‘BACK TO SEARCH’ button. I have put this message in red text so it draws the users attention and is clear to users why a set has not been displayed.

#### Piece Checklist Page

Table

Description automatically generated

Figure 14: Piece Checklist Page

Diagram

Description automatically generated with medium confidence

Figure 15: Exit Checklist without Saving Popup

#### Image Popups

A picture containing shape

Description automatically generated

Figure 16: Popups for Piece Images and Set Images

Figure 16 shows two popups both of which display images. The top popup displays the image of a Lego Piece and the bottom popup displays the image of a Lego Set. These popups are opened when a picture of a Lego Piece or Lego Set is clicked on. These popups are used so that if the image on the page is small, users can expand it so they can see it easier. The ‘X’ in the top right corner of both popups dismisses them.

#### Import Checklist Popup

Graphical user interface

Description automatically generated with low confidence

Figure 17: Import Checklist Popup

The import checklist popup (Figure 17) is used to import a Lego Set Piece checklist CSV file that has been previously exported, and is opened via clicking the ‘Import Checklist’ link on the navbar of the Piece Checklist Page (Figure 14). Clicking the ‘Chose File’ button will open the file browser on the system the user is viewing the website on, they can then select a file to upload to the website.

The ‘Import’ button when clicked will try to import the selected file, if the file is valid (in the right format and a CSV file) the pieces stored will be imported and the user will be redirected to Figure 12: Lego Set Information Page. If they then click the ‘View Piece Checklist’ link on that page Figure 14: Piece Checklist Page will be opened showing the pieces of the imported set, with the piece quantities found matching those from the imported CSV file.

There are also three error alerts in Figure 17 in red boxes, these are displayed when the user clicks the ‘Import’ button but there is an error while importing. The top error alert is displayed if a file is not selected, the second alert is displayed if the CSV file is blank, and the third alert is displayed if the file is not a CSV file or not correctly formatted so cannot be read. Here I have used these alerts in the popup instead of a separate alert popup, as this makes it clear to the user that an error has occurred without the user having to spend time closing an alert popup breaking up their flow.

The ‘Cancel’ button and ‘X’ in the top right corner, both dismiss the popup.

#### Login and Sign-Up Popup

Graphical user interface, application

Description automatically generated

Figure 18: Login Popup

Graphical user interface, diagram

Description automatically generated

Figure 19: Sign Up Popup

#### Logout Popup

Graphical user interface, application

Description automatically generated

Figure 20: Logout Popup

The Logout Popup (see Figure 20 above) appears when a user clicks the ‘Logout’ link in the navbar on all pages (which is only visible when a user is logged in), this is used to confirm that the user wants to logout of their account and stops users logging out by misclicking. If the user clicks the ‘cancel’ button the popup is dismissed. If they click the ‘Yes’ button the right popup in Figure 20 is displayed, informing the user they have been logged out and with an ‘OK’ button that returns the user to the Home Page (see Figure 8: Home Page - logged out above).

#### Sets in Progress Page

Table

Description automatically generated

Figure 21: Sets in Progress - Filter Bar Open

Table

Description automatically generated

Figure 22: Sets in Progress - Sort Bar Open

Graphical user interface

Description automatically generated with medium confidence

Figure 23: Delete Set from Sets in Progress Popup

The Sets in Progress Page (Figure 21 and Figure 22) can be accessed by a logged in user clicking the ‘Sets in Progress’ link on the secondary navbar on the Home Page (Figure 9). On this page users can search their Sets In Progress to find the a Set they wish to continue the checklist for.

Similar as to the Search Page, in the centre of the page a list of Lego sets is displayed in rows, with columns displaying different bits of a Lego sets information. The blue square represents an image of a Lego Set that when clicked will open Figure 16 to view the image enlarged (see more detail above). There is a ‘plus’ icon button that when clicked opens a popup (Figure 26) to add a set to a setlist. There is also a ‘bin’ icon in each set row, that when clicked a popup appears (Figure 23) asking the user if they want to delete the selected set from their sets in progress, this is used so that users don’t accidentally delete a set from Sets in Progress by misclicking. There is a cancel button to dismiss the popup and a ‘Delete’ button that will delete the selected set from the Sets in Progress. The text ‘Set\_name’ is replaced with the name of the set that will be deleted.

Above the list of sets is a header that has the name for each column, so it is clear to the user what each of these columns contains. Each of these column names (excluding ‘Set Image’), has an arrow next to it, these show if the sets are being sort by that column, that works same as the Search Page above.

Clicking one of these arrows will sort the sets in the column the arrow is adjacent to. If a down arrow is clicked the sets will be sorted descending and the arrow will change to an up arrow, if it’s an up arrow they will be sorted ascendingly, and the arrow will change to a down arrow. If an up-down arrow is clicked the sets will be sorted ascendingly, and this arrow will change to a down arrow and any other arrows that are not up-down arrows will change to up-down as those columns are no longer being filtered. Clicking on an arrow to sort a list will remove any other sorts currently active including a multi-sort selected via the sort bar, which is explained below.

There is a scroll bar down the side of the page so users can scroll through this list of Lego Sets.

If the user clicks a Sets set number, which is a hyper link, Figure 12: Lego Set Information Page will open displaying the selected set. If they then click the ‘View Piece Checklist’ link on that page Figure 14: Piece Checklist Page will be opened showing the saved progress of the Set, with the piece quantities found matching those saved previously.

At the top of the page is a black navbar containing the websites name so it’s clear and obvious to users what website they are on, that when clicked acts as a link returning the user to the Home Page (4.6.1). Next to this is some text informing the user they are viewing their Sets in Progress. Then there are 2 number boxes where users can enter a Lego set number and variant number (at least 1) and a button that will search for the Lego Set and return Figure 12 or Figure 13, these are explained in 4.6.3 Set Page above. Finally, there is a ‘Logout’ navbar link which opens Figure 20: Logout Popup, these are both explained above (in sections 4.6.7 and 4.6.8 respectively).

There is a navbar link ‘Filter’ which opens a filter bar underneath the navbar (shown open in Figure 21) and functions the same as on the Search Page but filtering a user’s Sets in Progress instead of sets from the Rebrickable API [1].

(shown open in Figure 10), closing the sort bar (shown open in Figure 11) if it is open. Likewise, the ‘Sort’ navbar link opens a sort bar underneath the navbar shown open in Figure 11, closing the filter bar if it is open.

The filter bar (Figure 10) contains filters the user can use to filter the Lego Sets, there is a text box to search by text (e.g. ‘Clone’), a select box to filter by Lego Set Theme (e.g. Star Wars, Lego City, Batman), ‘Minimum Year’ and ‘Maximum Year’ number boxes that filter between these years, ‘Minimum Pieces’ and ‘Maximum Pieces’ number boxes that filters the sets by the minimum and maximum number of Lego Pieces in the set and the ‘Filter’ button then applies these filters to all Lego Sets (via a call to the Rebrickable API [1]).

The sort bar (Figure 22) contains three select boxes so a user can perform a multi-sort, which functions the same as on the Search Page but sorting a user’s Sets in Progress instead.

containing all the columns sorts ascending and descending with also the option of none. The first select must always be selected, and by default the sets are sorted by ‘Set Number’. This is used selecting a column to sort by and then the preceding column to sort by, up to a maximum of three columns, this means where the first column values of sets match then they are compared by the second column value instead e.t.c. There is then a ‘Sort’ button that when pressed applies this sort to the list.

For example, in Figure 11 the sets are first being filtered by ‘Year Released’ ascending and then by ‘Set Number’ ascending, so the ‘Set 53’ and ‘Set 54’ both have ‘Year Released’ of 2001 so they are then compared by ‘Set Number’ and as ‘Set 53’ has a set number of ’53-1’ and ‘Set 54’ has a set number of ’54-1’ ‘Set 53’ appears before ‘Set 54’.

The breadcrumb trail, also in the footer, has the current page in white text, with the previous pages to the right with hyperlinks to those pages. I added this so a user can see their position on the user journey.

#### Set Lists Page

Table

Description automatically generated

Figure 24: Set Lists Page

#### Create New Set List Popup

Graphical user interface, application

Description automatically generated

Figure 25: Create New Set List Popup

#### Add Set to a Set List Popup

Table

Description automatically generated

Figure 26: Add Set to Set List Popup

The Add Set to a Set List Popup (Figure 26) is displayed when a logged-in user clicks the ‘plus’ icon button on the Search Page, Sets in Progress Page, Set List Page or clicks the ‘Add Set to List’ button on the Set Page. Using this popup users can add a Lego Set to one of their setlists. The text ‘set\_number’ is replaced with the set number of the set the user wants to add to a list.

Using the select dropdown, the user can select which of their setlists to add the set too. The ‘Add Set’ button adds the set to the selected list and closes the popup.

The ‘Cancel’ button and ‘X’ in the top right corner both dismiss the popup, and the ‘New List’ button opens the Create New Set List Popup (Figure 25) on top of this popup, where users can create a new setlist to add the set to (for more detail see 4.6.11 Create New Set List Popup above).

#### Set List Page

Table

Description automatically generated

Figure 27: Set List - Filter Bar Open

Table

Description automatically generated

Figure 28: Set List - Sort Bar Open

Graphical user interface

Description automatically generated with medium confidence

Figure 29: Delete Set from Set List Popup

Set list Page info.

When the ‘bin’ icon in a set row is clicked, a popup (Figure 29) appears asking the user if they want to delete the selected set from the setlist, this is used so that users don’t accidentally delete a set from a list by misclicking. There is a cancel button to dismiss the popup and a ‘Delete’ button that will delete the selected set from the setlist. The text ‘Set\_name’ and ‘setList\_name’ are replaced with the name of the set that will be deleted and the name of the list it will be deleted from respectively.

#### Edit Set List Popup

Graphical user interface, table

Description automatically generated with medium confidence

Figure 30: Edit Set List Popup

When the ‘edit’ icon is clicked on a setlist row on Figure 24: Set Lists Page, or the ‘edit’ link in the navbar on the Set List Page (4.6.13) is clicked, a popup (Figure 30) appears where a user can enter a new name for their setlist. There is a cancel button to dismiss the popup and a ‘Save’ button that will save the selected setlist with the new name. The text ‘list\_name’ is replaced with the name of the setlist that will be deleted.

#### Delete Set List Popup

Graphical user interface, text

Description automatically generated

Figure 31: Delete Set List Popup

When the ‘bin’ icon is clicked on a setlist row on Figure 24: Set Lists Page, or the ‘delete’ link in the navbar on the Set List Page (4.6.13) is clicked, a popup (Figure 31) appears asking the user if they want to delete the selected setlist, this is used so that users don’t accidentally delete a list by misclicking. There is a cancel button to dismiss the popup and a ‘Delete’ button that will delete the selected setlist. The text ‘list\_name’ is replaced with the name of the setlist that will be deleted.

#### Profile Page

Graphical user interface, application

Description automatically generated

Figure 32: Profile Page - Password Set, Google Account Not Linked

Graphical user interface

Description automatically generated

Figure 33: Profile Page - Password Not Set, Google Account Linked

#### Change Email Popup

Graphical user interface, diagram

Description automatically generated

Figure 34: Change Email Popup

#### Change Password Popup

Graphical user interface, diagram

Description automatically generated

Figure 35: Change Password Popup

#### Delete Account Popup

Graphical user interface, text

Description automatically generated

Figure 36: Delete Account Popup

When the ‘Delete your account’ link is clicked on the Profile Page (4.6.16 above), a popup (Figure 36) appears asking the user if they are sure they want to delete their account, with a tick box that must be ticked to active the ‘Delete’ button so the user can delete their account. This popup is used so that users don’t accidentally delete their account by misclicking and the tick box ensures the user understands they are deleting their account. There is also a cancel button to dismiss the popup.

## Implementation

## Results and Discussion

## Critical Appraisal

## Conclusion

## References

1. "Rebrickable API | Rebrickable - Build with LEGO", *Rebrickable.com*. [Online]. Available: https://rebrickable.com/api/. [Accessed: 17- Nov- 2021]
2. "Rebrickable API Documentation | Rebrickable - Build with LEGO", *Rebrickable.com*, 2020. [Online]. Available: https://rebrickable.com/api/v3/docs/?key=15b84a4cfa3259beb72eb08e7ccf55df. [Accessed: 15- Nov- 2021]
3. "BrickLink - Buy and sell LEGO Parts, Sets and Minifigures", *Bricklink.com*. [Online]. Available: <https://www.bricklink.com/v2/main.page>. [Accessed: 17- Nov- 2021]
4. "Rebrickable | Rebrickable - Build with LEGO", *Rebrickable.com*. [Online]. Available: https://rebrickable.com/. [Accessed: 17- Nov- 2021]
5. "LEGO Set 75280-1 501st Legion Clone Troopers (2020 Star Wars) | Rebrickable - Build with LEGO", *Rebrickable.com*, 2021. [Online]. Available: https://rebrickable.com/sets/75280-1/501st-legion-clone-troopers. [Accessed: 17- Nov- 2021]
6. "LEGO PART 6134 Hinge Brick 2 x 2 Top Plate Thin | Rebrickable - Build with LEGO", Rebrickable.com. [Online]. Available: https://rebrickable.com/parts/6134/hinge-brick-2-x-2-top-plate-thin/#buy\_parts. [Accessed: 24- Feb- 2022] Huw, "API version 3 documentation", *Brickset.com*, 2021. [Online]. Available: https://brickset.com/article/52664/api-version-3-documentation. [Accessed: 17- Nov- 2021]
7. Mihalcea, V., Ebersole, S., Boriero, A., Morling, G., Badner, G., Cranford, C., Bernard, E., Grinovero, S., Meyer, B., Ferentschik, H., King, G., Bauer, C., Andersen, M., Maesen, K., Vansa, R. and Jacomet, L., n.d. Hibernate ORM 5.5.9.Final User Guide. [online] Docs.jboss.org. Available at: https://docs.jboss.org/hibernate/orm/current/userguide/html\_single/Hibernate\_User\_Guide.html [Accessed 28 February 2022].
8. del

## Appendix A

Questionnaire for what user would want from a digital checklist for pieces in a Lego Set.Text, letter

Description automatically generated

Table

Description automatically generatedTable

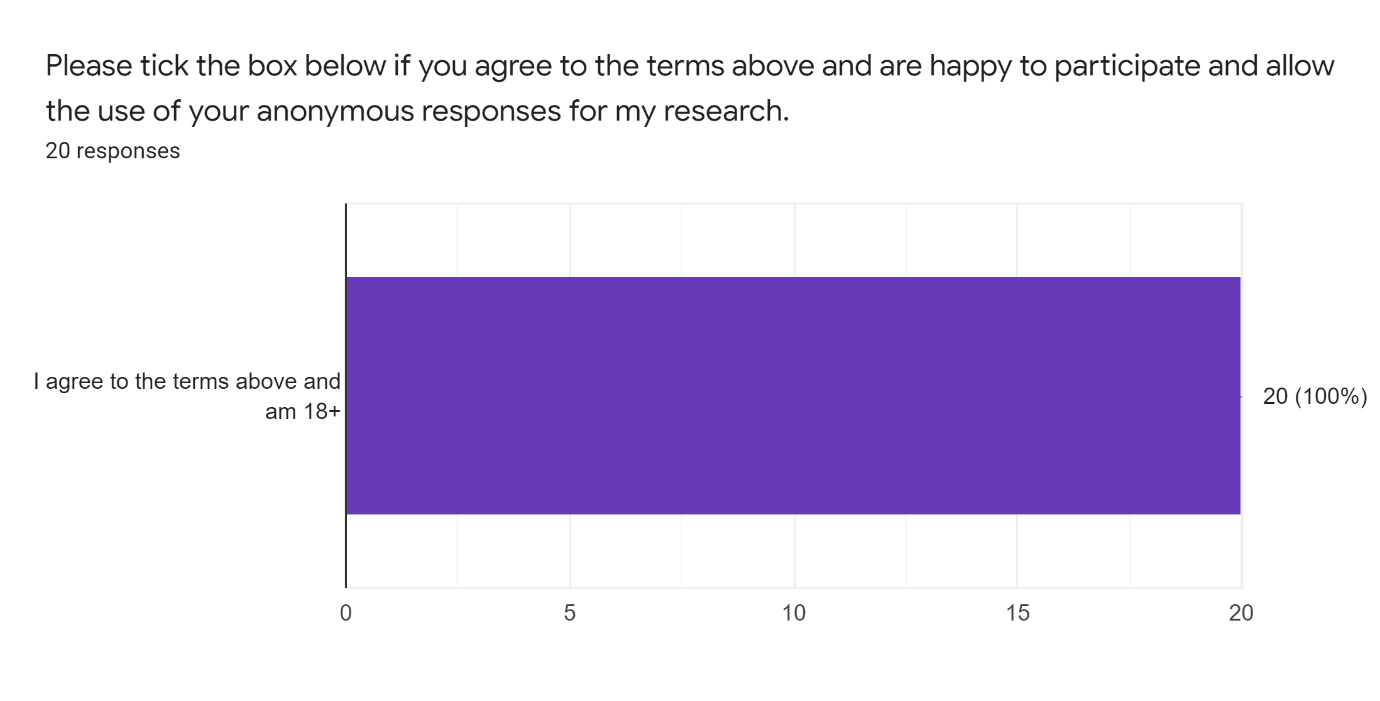
Description automatically generatedText

Description automatically generated

## Appendix B

Results from my questionnaire (see **Appendix A**).

Letter of Consent Result



Chart, pie chart

Description automatically generated

Chart, pie chart

Description automatically generatedGraphical user interface, text, application

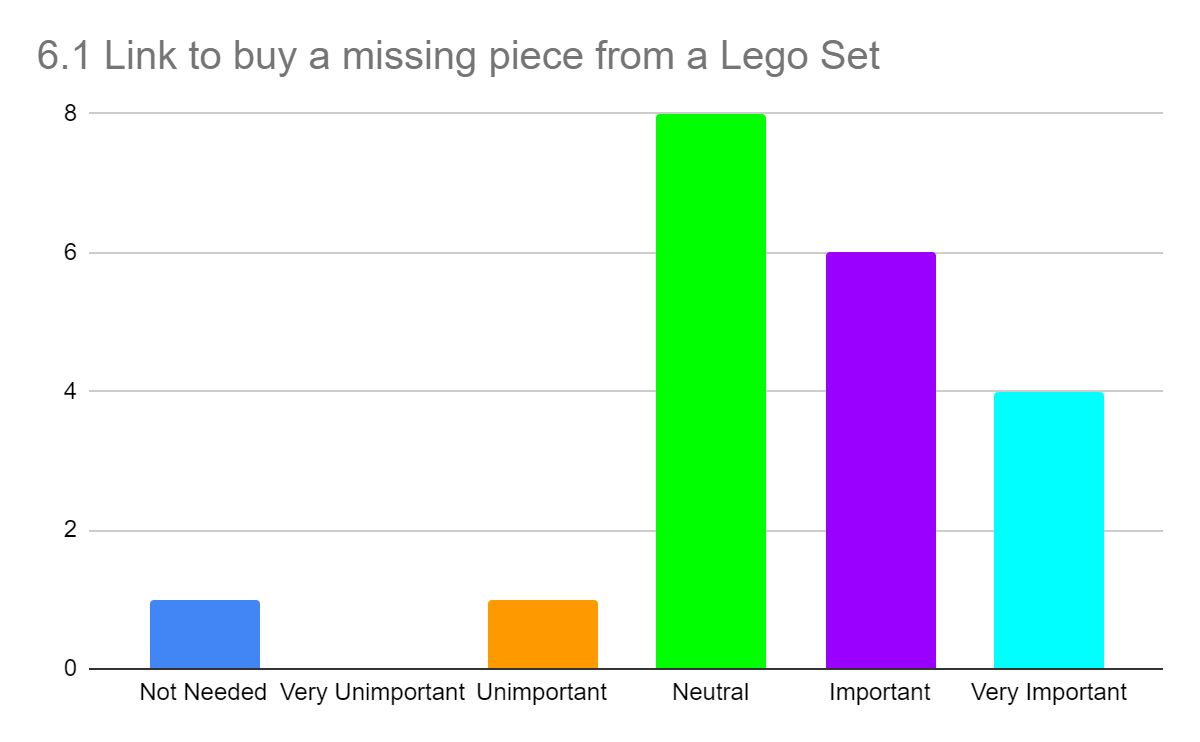
Description automatically generatedChart, bar chart

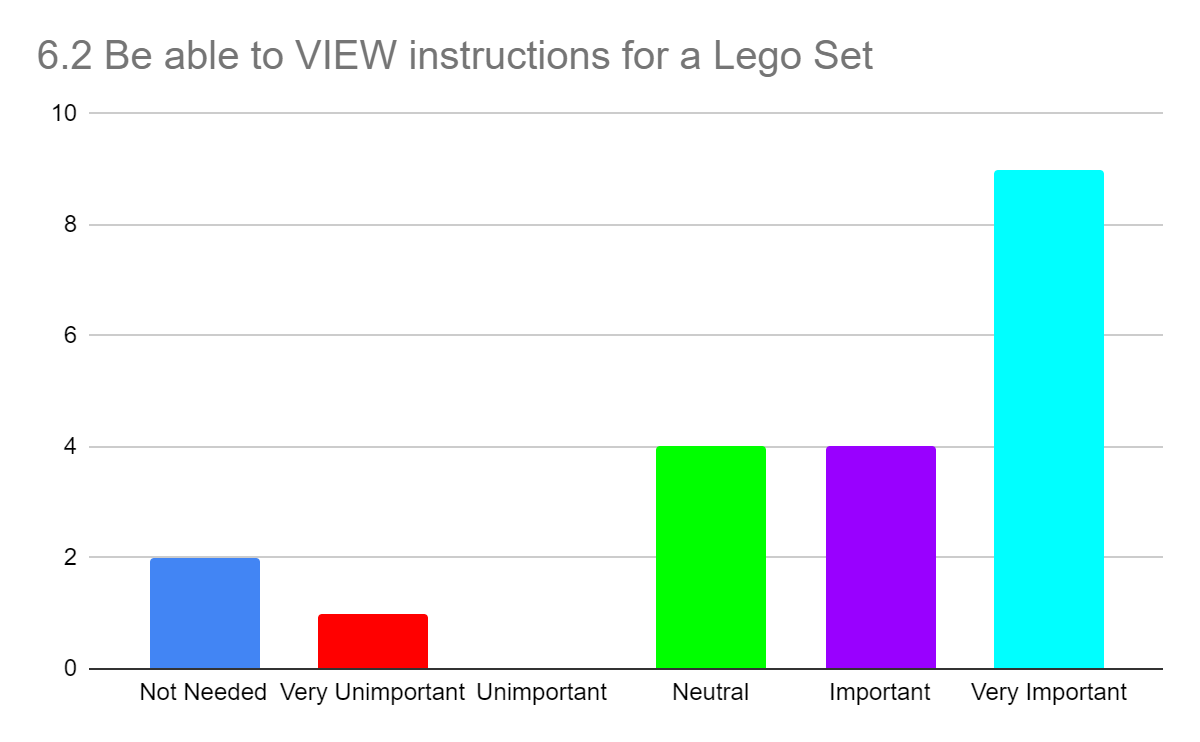
Description automatically generated

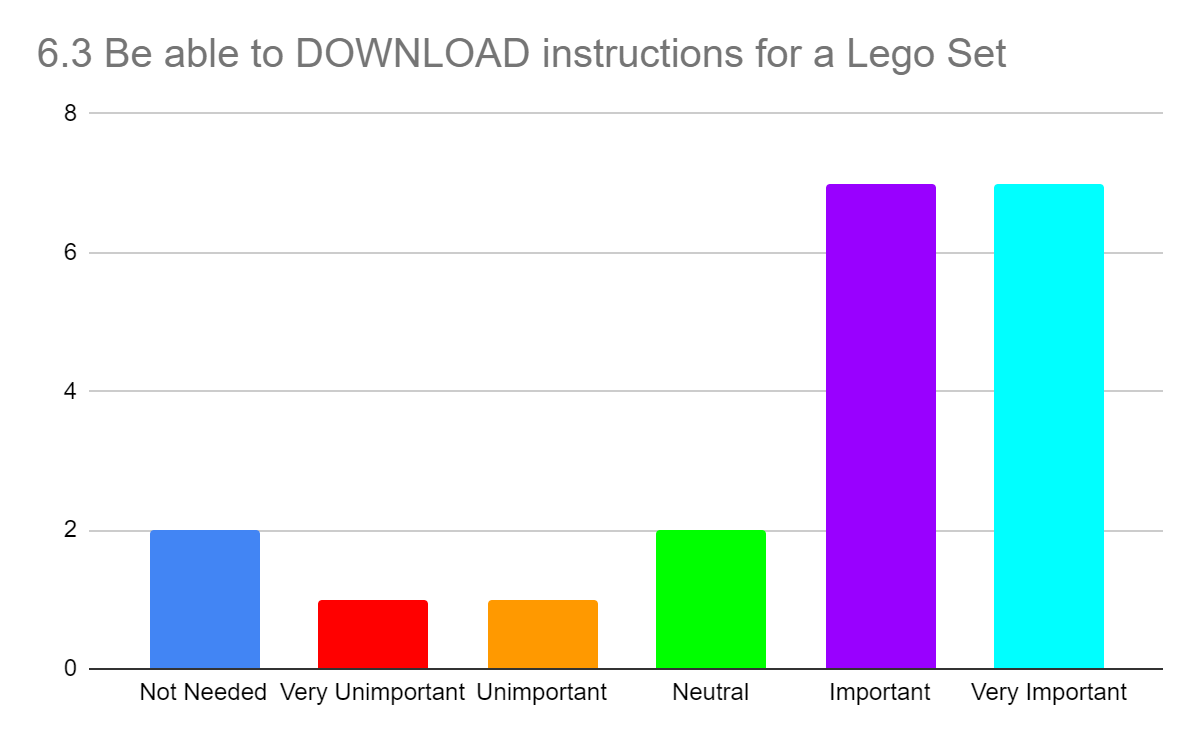
Chart, bar chart

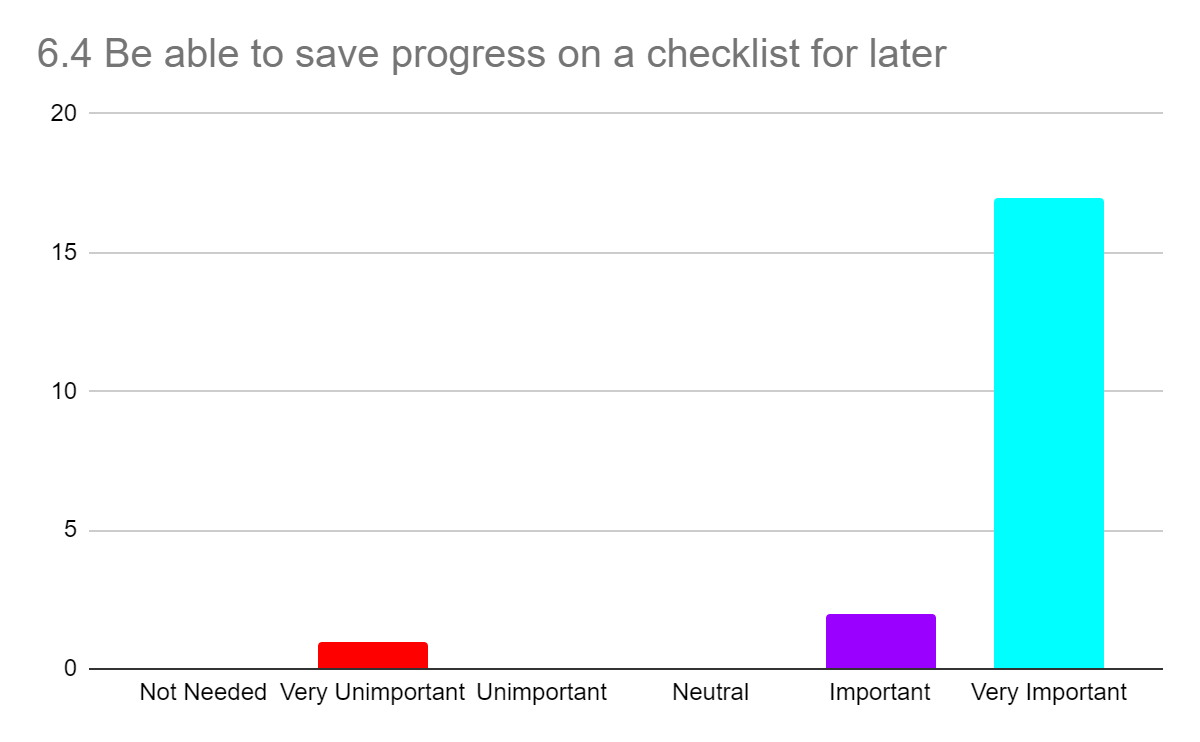
Description automatically generated

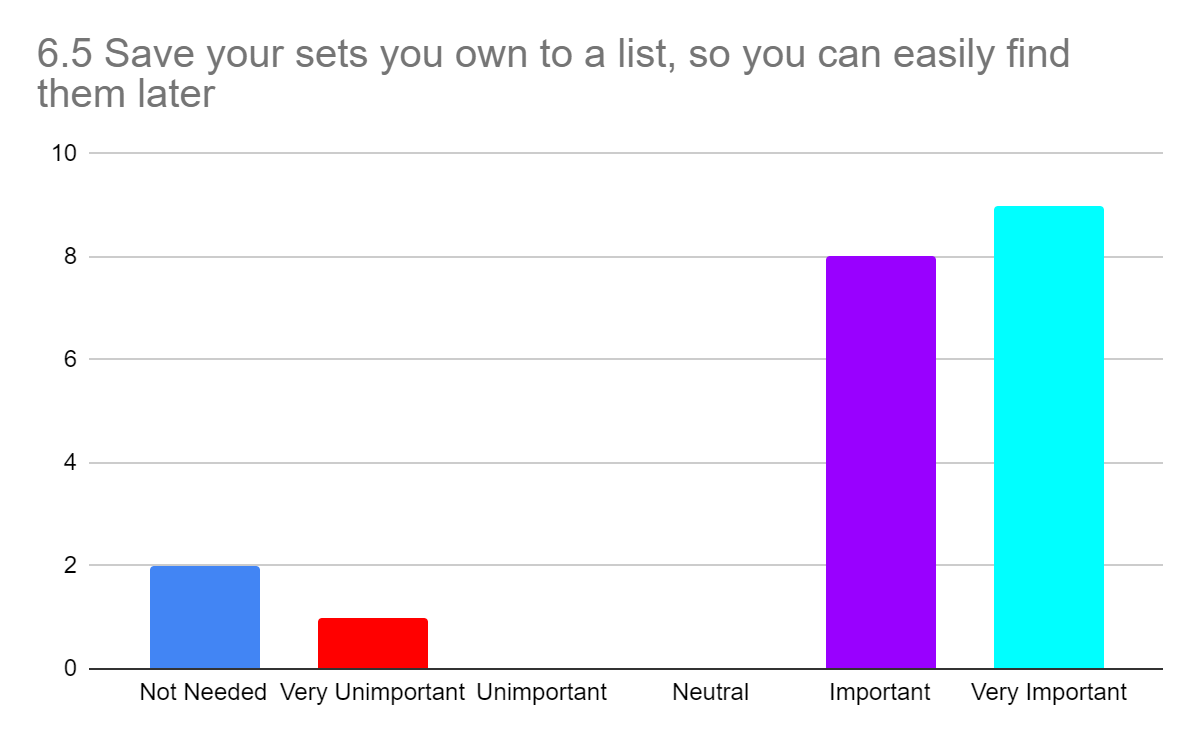
6. How important would the following features be to you in a Digital Checklist for Pieces in a Lego Set ?

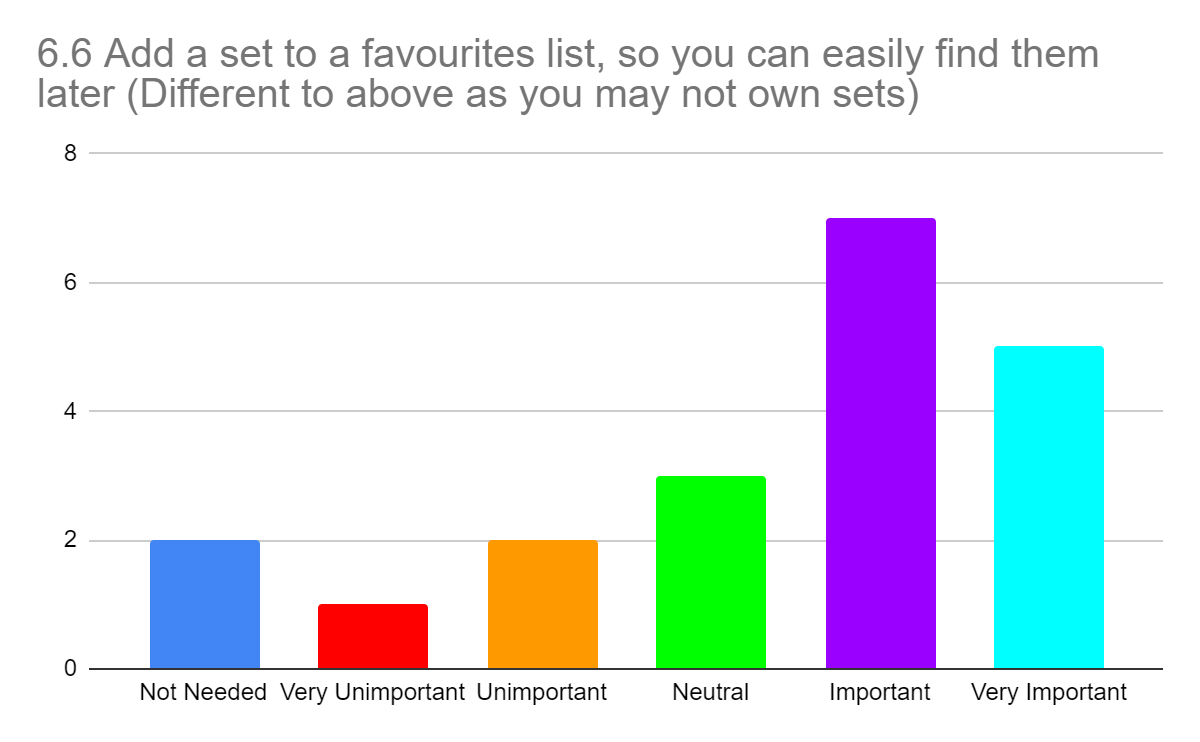












Graphical user interface, text, application, email

Description automatically generated