School of Computing and Mathematical Sciences, University of Leicester

CO3201 Computer Science Project: Interim Report

Lego: Set Checklist Creator

DANiel Richard Booth

2021

Table of Contents

[Declaration ii](#_Toc88670202)

[Aims and Objectives 1](#_Toc88670203)

[1.1 Introduction 1](#_Toc88670204)

[1.2 Aim 1](#_Toc88670205)

[1.3 Objectives 1](#_Toc88670206)

[Survey of Literature/Information Sources 1](#_Toc88670207)

[Requirements 4](#_Toc88670208)

[1.4 Key Features: 4](#_Toc88670209)

[1.5 Nice to have Features: 4](#_Toc88670210)

[1.6 Optional Features 4](#_Toc88670211)

[Outline of Specification and Design 5](#_Toc88670212)

[1.7 High-level overview of the architecture of the system 5](#_Toc88670213)

[1.8 APIs 5](#_Toc88670214)

[1.9 Database 6](#_Toc88670215)

[Planning and Timescales 7](#_Toc88670216)

[1.10 Semester 1: 08/10/2021 – 14/01/2022 7](#_Toc88670217)

[1.11 Semester 2: 17/01/2022 – 09/05/2022 8](#_Toc88670218)

[References 8](#_Toc88670219)

[Appendix A 9](#_Toc88670220)

[Appendix B 13](#_Toc88670221)

## Declaration

Text, letter

Description automatically generated

## Aims and Objectives

### 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, over time 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

Challenges I expect to face during the project are

* Linking a Lego Set API to a program
* Transforming data from a JSON file into a class
* Developing a cross-platform website

## Survey of Literature/Information Sources

To begin with, I looked up the Rebrickable API [2] 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 [3], 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.

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.

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

Question 1 showed me that the majority of users (75% see **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 [4]. 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 [5]. 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 [6]) 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.

The results of question 4 (see **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.

Question 6 results (see **Appendix B**) shows that it is very important to most people (17 people) 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 [7] 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.

Finally, the results of question 7 (see **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 [4] wanted list. I took these suggestions into account when designing my requirements.

## 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 [2]
* 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
* 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 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 XML

## Outline of Specification and Design

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

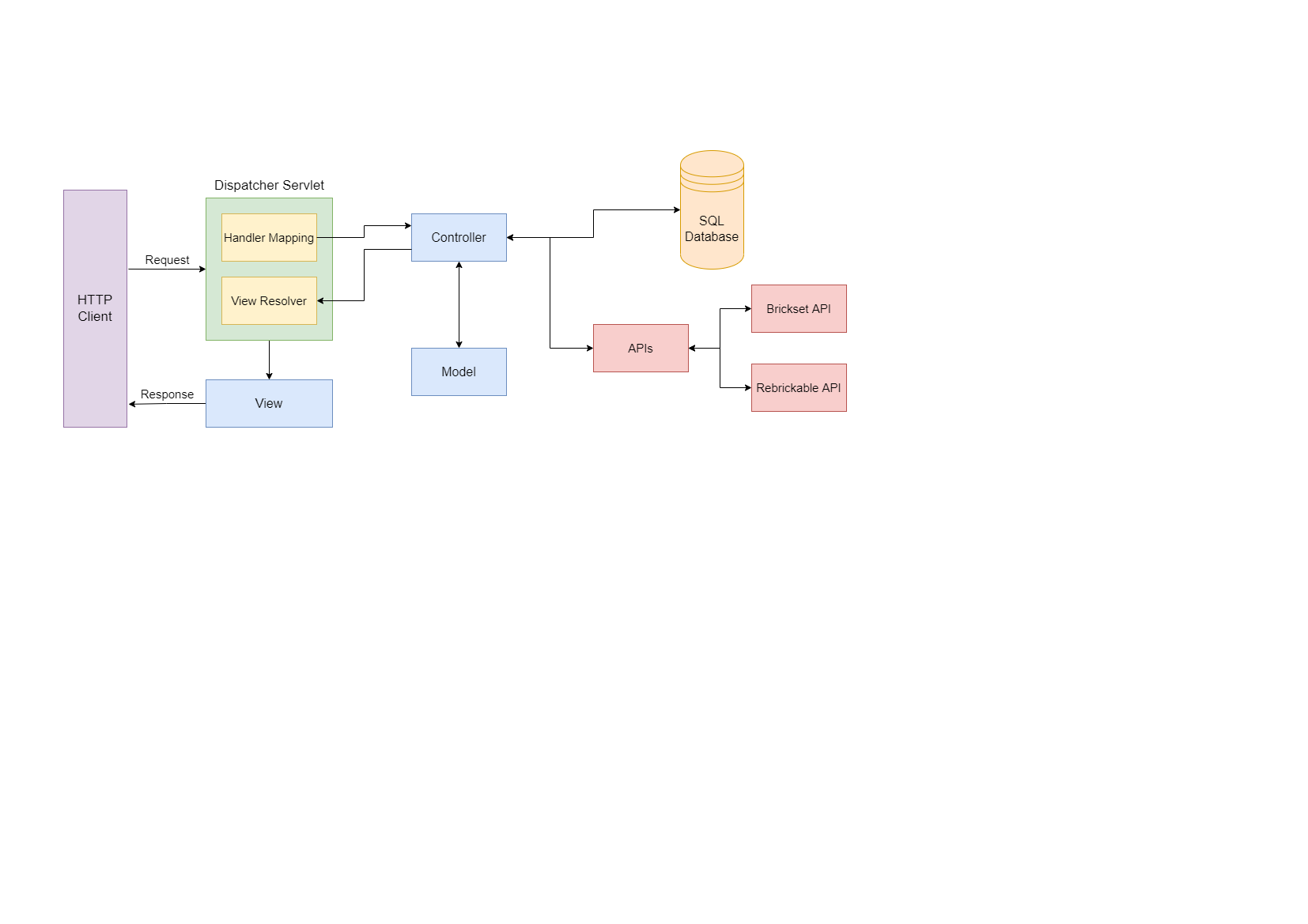


Figure 1 – High-Level Overview

Figure 1 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 [2] (as mentioned above see 0 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 [7] (as mentioned above, see 0 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

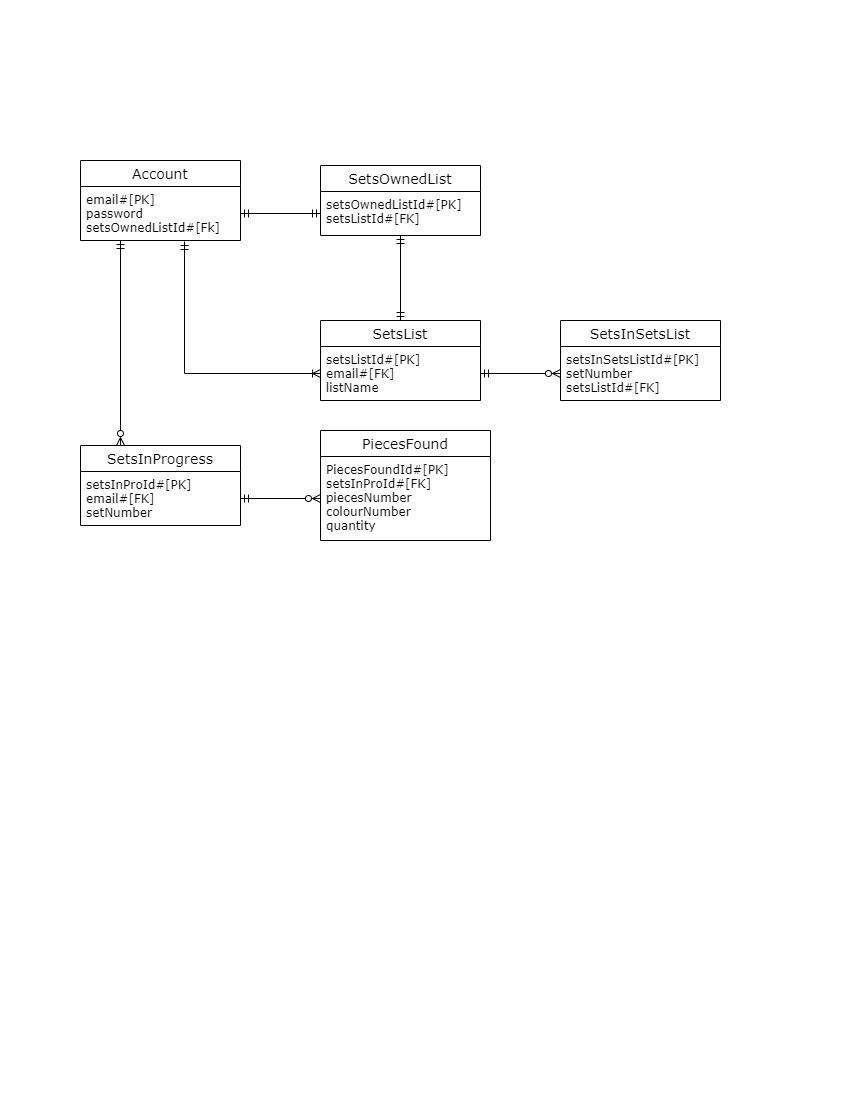


Figure 2 – ER Class Diagram

I will use MySQL for my database, and Figure 2 above shows an ER class diagram for this database, that may change depending on the requirements completed. The database will have several tables called Account, SetsInProgress, PiecesFound, SetsList, SetsInSetsList and SetsOwnedList.

Account will be used to store user accounts, with their email as the primary key (as this is already unique), a password (that will be encrypted by hashing with salt) and a foreign key setsOwnedListId that links to the SetsOwnedList table which is explained later in this section.

SetsInProgress stores the sets that a user currently has a checklist in progress for, with the user’s email as the foreign key so it's easy to find which user it belongs to and the set number. Then the pieces that the already been found are stored in the PiecesFound table, which stores the pieces number, colour number and the quantity currently found (if 0 pieces have been found these won’t be saved to save database storage).

SetsList table will store lists the user has created, that contain Lego sets (called set lists). The table stores the name of the list and a unique id of the list that is used to find the sets that belong to the list in the SetsInSetsList table. SetsInSetsList stores the set numbers of a set saved to a list, with a unique id as the primary key for each set saved, a set number should not be used a set could be in more than one list.

Finally, when a user creates an account, they are automatically given an empty set list called Sets Owned List, and this list is stored in the SetsList table with other set lists. This list is linked to a user via the SetsOwnedList table that holds the id of the row in the SetsList table and a unique number called setsOwnedListId that (as mentioned above) is also a foreign key in the Account table. This is done so that a user’s Set Owned List can be easily found.

## Planning and Timescales

### Semester 1: 08/10/2021 – 14/01/2022

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Start Date | End Date | Status |
| Create list of possible features | 08/10/2021 | 21/10/2021 | Completed |
| Create basic project timeline | 08/10/2021 | 21/10/2021 | Completed |
| Research | 22/10/2021 | 26/11/2021 | In Progress |
| Read API Documentation and Test Features | 22/10/2021 | 22/10/2021 | Completed |
| Preliminary Research on Similar Projects | 25/10/2021 | 26/10/2021 | Completed |
| Feature Questionnaire | 27/10/2021 | 22/11/2021 | Completed |
| Design and Create Feature Questionnaire | 27/10/2021 | 27/10/2021 | Completed |
| Keep Sending off Questionnaire for Feedback and make changes | 28/10/2021 | 12/11/2021 | Completed |
| Ethics Consent for Question | 15/11/2021 | 15/11/2021 | Completed |
| Send off and Wait for Results (let run over weekend) | 16/11/2021 | 19/11/2021 | Completed |
| Analysis Questionnaire Results and Decide Features | 22/11/2021 | 22/11/2021 | Completed |
| Further Research on Similar Projects | 28/10/2021 | 01/11/2021 | Completed |
| Research Spring MVC and how to connect to an API | 25/11/2021 | 26/11/2021 | In Progress |
| Interim Report Write-up | 15/11/2021 | 25/11/2021 | Completed |
| Interim Report Deadline | 25/11/2021 | 25/11/2021 |  |
| Create Basic Spring MVC Program | 29/11/2021 | 30/11/2021 | Not Started |
| Link Program to Rebrickable API | 01/12/2021 | 01/12/2021 | Not Started |
| Add basic set number search using API and Display JSON as result | 02/12/2021 | 02/12/2021 | Not Started |
| Research how to transform a JSON file into a Java Class | 03/12/2021 | 07/12/2021 | Not Started |
| Implement a class that turns a Lego Set JSON result into a class | 08/12/2021 | 10/12/2021 | Not Started |
| Implement a class that turns a Lego Piece JSON result into a class | 13/12/2021 | 15/12/2021 | Not Started |
| Implement a feature that turns each Lego Piece in JSON list into instance of Lego Piece Class | 16/12/2021 | 17/12/2021 | Not Started |
| Add feature to display all pieces in a Lego set with an image | 20/12/2021 | 22/12/2021 | Not Started |
| Add Basic Check feature to list of pieces in a set | 23/12/2021 | 28/12/2021 | Not Started |
| Add feature to save checklist progress to device storage | 29/12/2021 | 03/01/2022 | Not Started |
| Improve search feature so that their text search that displays possible list of sets with image | 04/01/2022 | 07/01/2022 | Not Started |
| Add filter and sort by theme and year made to search feature | 10/01/2022 | 12/01/2022 | Not Started |
| Make improvements to website UI | 13/01/2022 | 14/01/2022 | Not Started |

Gantt charts

### Semester 2: 17/01/2022 – 09/05/2022

## References

1. Del
2. "Rebrickable API | Rebrickable - Build with LEGO", *Rebrickable.com*. [Online]. Available: https://rebrickable.com/api/. [Accessed: 17- Nov- 2021]
3. "Rebrickable API Documentation | Rebrickable - Build with LEGO", *Rebrickable.com*, 2020. [Online]. Available: https://rebrickable.com/api/v3/docs/?key=15b84a4cfa3259beb72eb08e7ccf55df. [Accessed: 15- Nov- 2021]
4. "BrickLink - Buy and sell LEGO Parts, Sets and Minifigures", *Bricklink.com*. [Online]. Available: <https://www.bricklink.com/v2/main.page>. [Accessed: 17- Nov- 2021]
5. "Rebrickable | Rebrickable - Build with LEGO", *Rebrickable.com*. [Online]. Available: https://rebrickable.com/. [Accessed: 17- Nov- 2021]
6. "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]
7. Huw, "API version 3 documentation", *Brickset.com*, 2021. [Online]. Available: https://brickset.com/article/52664/api-version-3-documentation. [Accessed: 17- Nov- 2021]
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

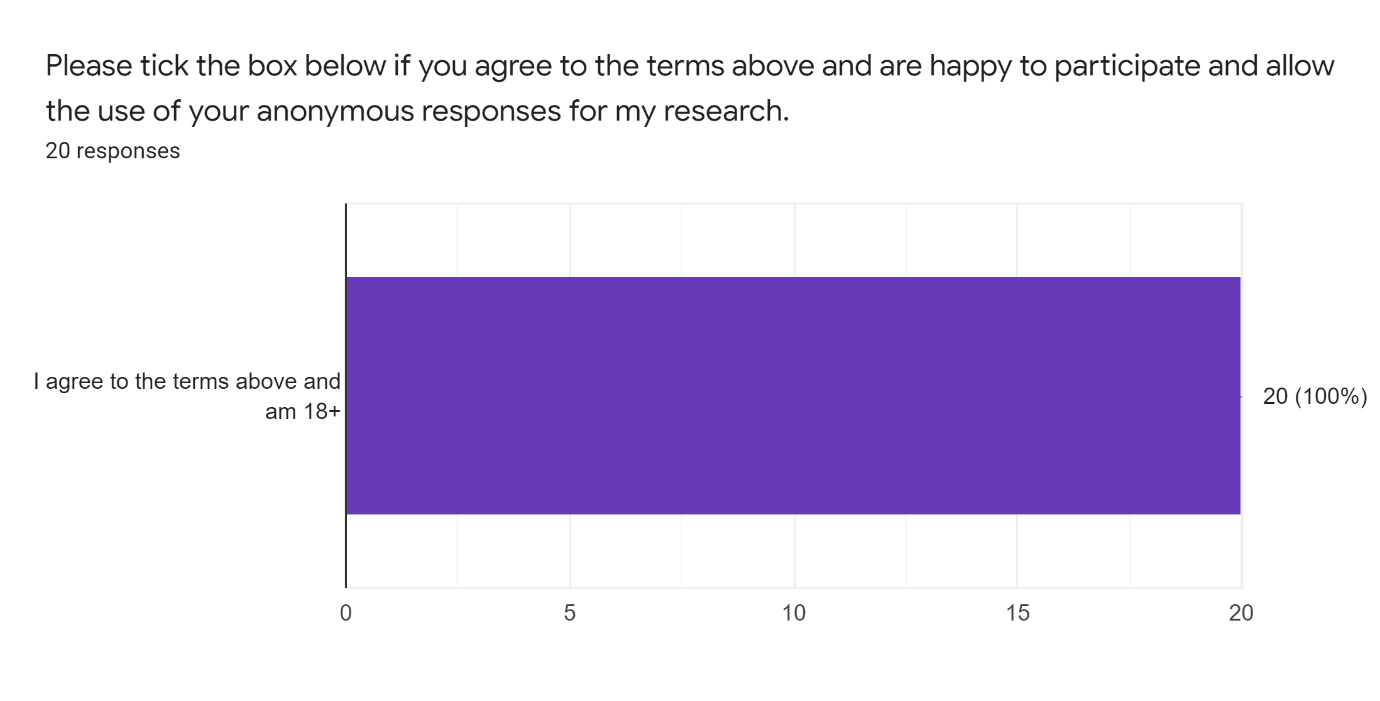
Description automatically generatedText

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

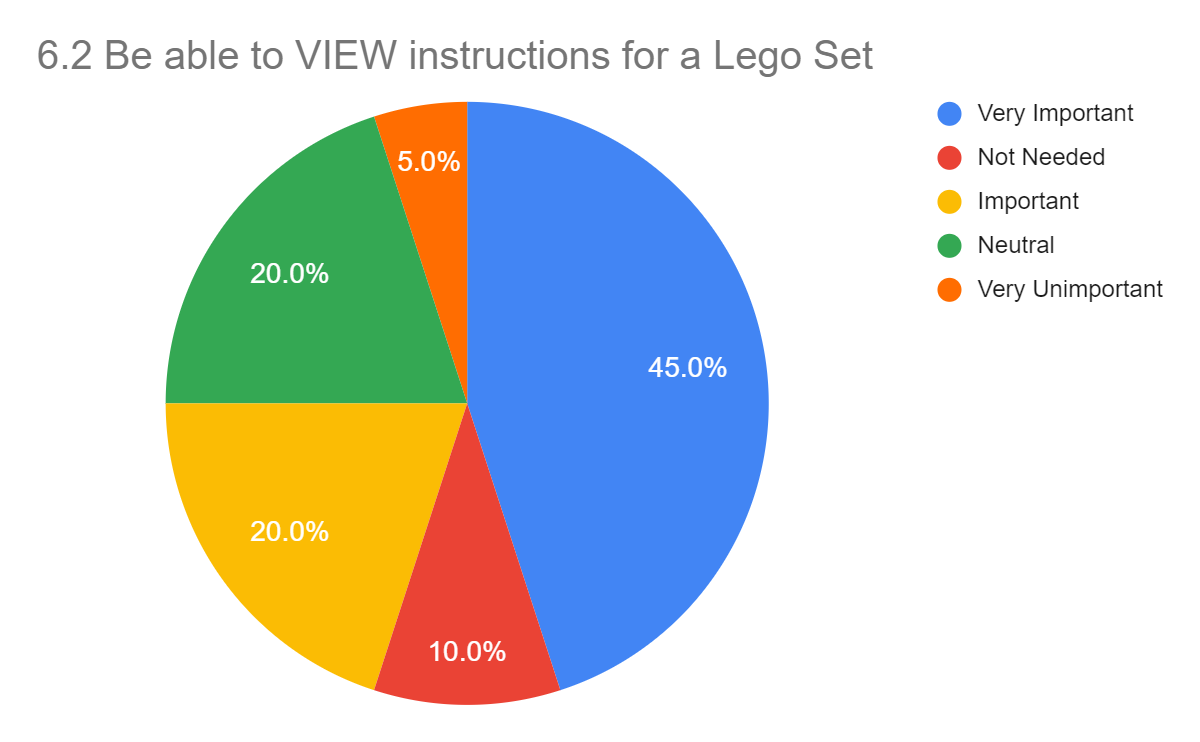
Description automatically generated

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 ?

Chart, pie chart

Description automatically generated

Chart, pie chart

Description automatically generated

Chart, pie chart

Description automatically generated

Chart, pie chart

Description automatically generated

Chart, pie chart

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated