

[8] Don Ho. (Last updated August 30 2017). NOTEPAD++.

Available: <https://notepad-plus-plus.org/>

Last accessed 30 sep 2017.

[9] Sony. (Original release May 2016). SONY XPERIA X PHONE SPECIFICATIONS.

Available: <https://www.sonymobile.com/global-en/products/phones/xperia-x/specifications/>

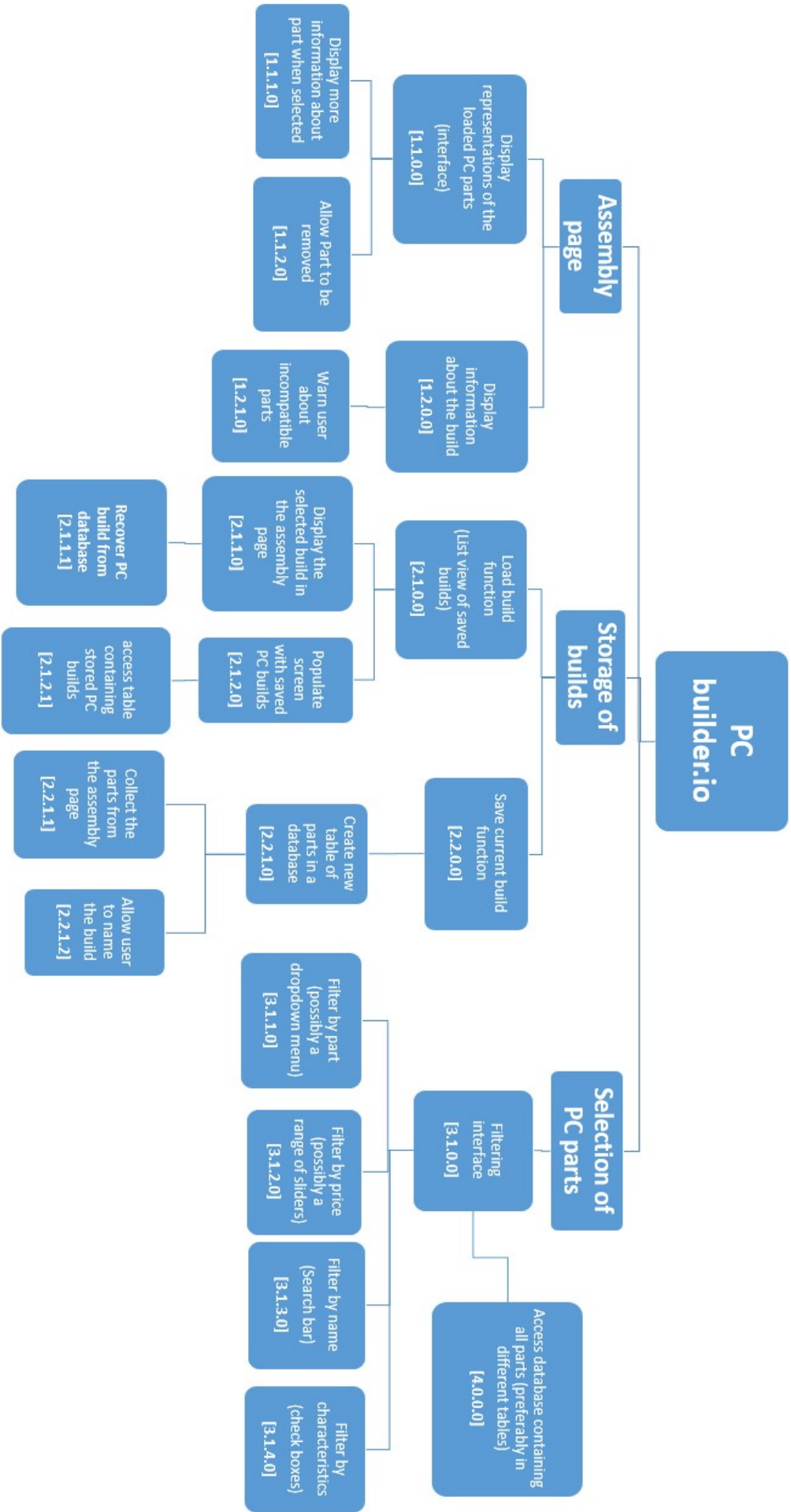
Last accessed 30 sep 2017.

Design

In order to design and go on to create a solution, I have gone through a process of decomposition of the problem and ways to solve it. My decomposition approaches the solution from the perspective of the content of 3 pages in the application that resolve different aspects of the problem to provide a solution. The initial layers will be for the user interface and the deeper layers the tree the for the problems in the 'backend' of the program (i.e. file & database management and source code).

Decomposition tree

Tree with annotated explanations



Decomposition module name [reference number]	Annotation
Assembly page [1]	This is where the current build is displayed and the user can look at here selected parts and information about the build. All the features of this page is referenced with [1.X.X.X]
Display representations of the loaded PC parts (interface) [1.1.0.0]	In this module the app should be able to graphically (on the user interface) represent each apart on the screen. This module and its sub modules are to resolve the “application must be able to display parts in the PC build” success criteria. This can be done using images as sprites. This will allow the user to identify the parts that they have in the current build and a comparison to what they look like in real life for when they want to build the PC.
Display more information about part when selected [1.1.1.0]	For a better user experience, they should be able to see more about the parts they have selected from the assembly. This is so they can judge whether or not they like the parts they have selected.
Allow Part to be removed [1.1.2.0]	In order for the user to experiment with parts they should be able to remove them from their current build as determined by the success criteria “The application must allow the user to remove parts from there build”. This can be done with a procedure (method) removes the record of the part from the build.
Display information about the build [1.2.0.0]	Another aspect of the assembly page is that it should provide the user with information about the build in general (e.g. the cost, required power etc.). This is good for user experience because it gives the user information for them to have a more informed decisions on the parts they want.
Warn user about incompatible parts [1.2.1.0]	In order to meet the success criteria of “The application must be able to recognise parts that are compatible with each other from parts that are not” it will need an algorithm that look at the socket standards for the parts and identify if they match, then inform the user. This warning of a compatibility miss match can be as part of [1.2.0.0]
Storage of builds [2]	The user should be able to manage the builds that they have stored to meet success criteria points “The application must be able to save PC builds” and “The application must be able to load saved PC builds” this page will allow users to do so. All the features of this page are referenced with [2.X.X.X]
Load build interface (List view of saved builds) [2.1.0.0]	This module is for the load function’s interface it is needed to allow the user to choose a build that they have been working on and continues to work on it. This interface can be a list view that shows the name of the builds saved in storage because it is a user friendly way of displaying items name (in this case PC names).
Display the selected build in the assembly page [2.1.1.0]	This module is needed to show the build has been selected from the interface [2.1.1.0] and the PC parts are loaded into the assembly page.

Recover PC build from database [2.1.1.1]	This select a build module is used to enable the interactivity to the Load build interface [2.1.0.0] and to recover the data from the database to pass on to the [2.1.1.0] module so the parts can be displayed on screen.
Populate screen with saved PC builds[2.1.2.0]	This module collects the names of the PC that have been stored. So that it can display the names on the list view in [2.1.0.0]. This can be an algorithm that allocates a section of the list view to a stored PC.
access table containing stored PC builds[2.1.2.1]	In order to populate the screen with the saved PC build names [2.1.2.0] the program must read the database. This can be done with an algorithm that returns the PC build names to [2.1.2.0] by this module.
Save current build function[2.2.0.0]	This module and its sub modules govern the save function's interface and the storage of the current PC build so that the user can continues experimenting at another time. I see this function being accessed by a button which would asked the user to enter PC build name and store the PC build.
Create new table of parts in a database[2.2.1.0]	This creates a storage format, I the form of a table to which the parts that should be store will be saved into and can be loaded back using [2.1.1.1]
Collect the parts from the assembly page[2.2.1.1]	The table in [2.2.1.0] will need to be populated with new parts. This module will populate the table with the parts from the assembly page [1] so that they can be stored.
Allow user to name the build[2.2.1.2]	This module will name the tables created by [2.2.1.0] this will be combined with the recovered parts from [2.2.1.1] to store the PC build. By naming the table the user can recognise which build they want to load.
Selection of PC parts [3]	This page allows the user to search the database for the parts they want to add to their build. This will meet the success criteria of "The application must allow the user to add PC parts to their build". All the features of this page are referenced with [3.X.X.X] and the database access assistant modules with [4.X.X.X].
Filtering interface[3.1.0.0]	The filtering interface will allow the user to enter a set of characteristics of the part they want to select and the program populates the screen with Items that fit the specified characteristics. The submodules of this ([3.1.X.X]) will be the different filtering option available of this page. Having these different filtering options allows us to meet the success criteria of "The application must allow the user to query the database in convenient ways".
Filter by part (possibly a dropdown menu)[3.1.1.0]	This module will enable the user to query the database by selecting the part they want. I can see this being done with a dropdown menu in which the sections correspond to a type of part (e.g. RAM, Motherboards etc.).
Filter by price (possibly a range of sliders)[3.1.2.0]	This module will enable the user to query the database by price using the sliders to set the minimum or maximum price of the parts they want the database to return to them.
Filter by name(Search bar)[3.1.3.0]	This module will enable the user to query the database by searching the name of the parts thy want to look at. I can see this being done with a search bar in which the text is matched with the names of the parts in the database (e.g. RAM, Motherboards etc.).

Filter by characteristics(check boxes)[3.1.4.0]	This module will enable the user to query the database by selecting the part they want. I can see this being done with a dropdown menu in which the sections correspond to a type of part (e.g. RAM, Motherboards etc.).
Access database containing all parts (preferably in different tables) [4.0.0.0]	This module will handle the retrieval of data using the filters. This will be done by converting the user inputs into an SQL query within this module.

Variables and data structures

As mentioned earlier the program is dependant of the database and the extraction and storing of information in the database. The database allows me to store information in an organised way; this is so that it can retrieve newly stored information after the application has be closed and restarted. The database format I am planning to used is uses the '.db' extension. This enables me to view the database outside the program for me to populate it and debug it using DB browser for SQL. I can also interact and query the database by using the SQLite3 plugin for the Unity game engine.

PC parts database

To store the range of PC parts that will be used to make the build I was split between having a unified, flat-file database containing all the parts or separating each category of part into standalone tables. This was because there will not be any use of foreign keys to reference other parts from the record of one part. However, I feel that using multiple standalone tables will enable me to speed up the search process for parts, which aids the user experience. This is because, more processor cycles are used to filter a long flat-file database of all the records (parts) to match the type of part the user wants, compared to the standalone tables, where the appropriate table containing the required parts can be accesses immediately. Standalone table also allow me to customise the field names and datatypes so that it better suites the type of parts being stored on in the table. This also avoids empty data cells where entities do not have properties that apply field. My application is in early prototyping so for now will only use four entities (records) for each table to prove that it works and I will design my code to be expandable and work with and unspecified amount of amount records that ca be added in the future.

From my speciation sections is have established that there are eight types of parts required to build a PC. Because of this, I will require eight tables I have shown my design for them below.

Database Filename: DBparts.db

Table Name: TBLmotherboard

Description: This table will be used to store a range of mother motherboards. Along with the name it will store the CPU socket, RAM socket, the amount of ram sockets, size standard and secondary data connection type. This is so compatibility checks can be made with other parts when it is selected. Finally, I will use a primary key to uniquely identify the motherboard when it is being searched for.

Field Name	Data Type	Description	Valid requirements	example
MOBO ID Primary key	String (can be converted to integer if required)	This is a unique number allocated to each motherboard (record) in the table. This is so each record can be identified when retrieving information about the selected part.	More than 0	"0001"
Name	String	This is the name of the motherboard	Not null	"Asus Prime Z370-P"
CPU Socket	String	This used to store the type of CPU socket that compatible with the motherboard. This is so the program	Not null	"LGA1151"

		can identify whether the selected CPU will work with the motherboard.		
RAM Socket	String	This is used to store the type of RAM stick that is compatible with the motherboard. This is so the program can identify whether the selected RAM stick will work with the motherboard.	Not null	"DDR3"
RAM Slots	Integer	This stores then amount of RAM sticks that can be attached to the motherboard.	Not null	2
Storage slots	Integer	This stores then amount of storage drives that can be attached to the motherboard.	More than 0 but less than or equal to 4	4
Form Factor	String	This stores the form factor of the motherboard, so it can be checked for compatibility with the selected case.	Not null	"Mini-ATX"
Storage port	String	This store the port type that can be used to attach a secondary storage device to the motherboard.	Not null	"SATA"
Price	Float	This stores the price of the motherboard. Measured in GBP(£).	More than 0, To two decimal places	45.99

Database Filename: DBparts.db

Table Name: TBLcpu

Description: This table will be used to store information about the CPUs.

Field Name	Data Type	Description	Valid requirements	example
CPU ID Primary key	String (can be converted to integer if required)	This is a unique number allocated to each CPU (record) in the table. This is so each record can be identified when retrieving information about the selected part.	More than 0	"0001"
Name	String	This is the name of the CPU.	Not null	"Intel I5-3740"
Clock speed	Float	This stores the clock speed of the CPU, so the user can compare and sort the CPUs by clock speed. It will be measured in gigahertz (GHz).	More than 0	3.4
Core count	Integer	This stores the amount of cores the CPU has, so the user can compare it to other CPUs	More than 0	2
Hyper-threads	integer	This stores the amount of threads the CPU has, so the user can compare it to other CPUs	More than 0	4

CPU Socket	String	This used to store the type of CPU socket the CPU is. This is so the program can identify whether the selected CPU will work with the selected motherboard.	Not null	"LGA1151"
Power demand	Float	This stores the amount of power required to run the CPU, so it can be added to the power budget to ensure the power requirements to run all the parts is met. Measured in watts (W).	More than 0	83.5
Price	Float	This stores the price of the CPU. Measured in GBP (£).	More than 0, To two decimal places	77.99

Database Filename: DBparts.db

Table Name: TBLpsu

Description: This table will be used to store information about the power supply units that can be used to power the builds.

Field Name	Data Type	Description	Valid requirements	example
PSU ID Primary key	String (can be converted to integer if required)	This is a unique number allocated to each power supply unit (record) in the table. This is so each record can be identified when retrieving information about the selected part.	More than 0	"0001"
Name	String	This is the name of the PSU	Not null	"Corsair RM750x"
Power rating	Float	This stores the maximum amount of power that the power supply unit can release, this is to ensure the power requirements to run all the parts is met. Measured in watts (W).	More than 0	550.00
Modular	Boolean	This store whether or not the power supply unit is modular. For the user to choose their preference.	Ether TRUE or FALSE	TRUE
Price	Float	This stores the price of the PSU. Measured in GBP (£).	More than 0, To two decimal places	94.99

Database Filename: DBparts.db

Table Name: TBLcpucooler

Description: This table will be used to store information about the CPU coolers that are available.

Field Name	Data Type	Description	Valid requirements	example
CPUcooler ID Primary key	String	This is a unique number allocated to each CPU cooler (record) in the table.	More than 0	"0001"

	(can be converted to integer if required)	This is so each record can be identified when retrieving information about the selected part.		
Name	String	This is the name of the CPU cooler	Not null	"Cooler Master Hyper 212 EVO"
Height	Float	This stores the height of the cooler to ensure it fits within the case. Measured in centimetres (cm).	More than 0	9.75
Noise level	Float	This stores the maximum expected noise level of the CPU cooler fan. This is so the user has an indication of what to expect from the cooler and can make a more informed choice. Measured in decibels (dbs).	more than 0	36.6
Power demand	Float	This stores the amount of power required to run the CPU cooler, so it can be added to the power budget to ensure the power requirements to run all the parts is met. Measured in watts.	More than 0	10.5
Price	Float	This stores the price of the CPU cooler. Measured in GBP (£).	More than 0, To two decimal places	48.28

Database Filename: DBparts.db

Table Name: TBLram

Description: This table will be used to store information about the RAM sticks that can be added the build

Field Name	Data Type	Description	Valid requirements	example
RAM ID Primary key	String (can be converted to integer if required)	This is a unique number allocated to each RAM stick (record) in the table. This is so each record can be identified when retrieving information about the selected part.	More than 0	"0001"
Name	String	This is the name of the RAM stick	Not null	"Kingston FURY"
RAM socket	String	This is used to store the type of RAM stick to ensure it is compatible with the selected motherboard.	Not null	"DDR3"
Memory size	Integer	This store the amount of memory in each stick of RAM so the user can sort by size and choose their preference. Measured in gigabytes (GB).	More than 0	4
bandwidth	Float	This stores the bandwidth of in each stick of RAM so the user can sort by bandwidth and choose their preference. Measured in gigahertz (Ghz).	More than 0	1.6

Power demand	Float	This stores the amount of power required to use the RAM, so it can be added to the power budget to ensure the power requirements to run all the parts is met. Measured in watts.	More than 0	18.0
Price	Float	This stores the price of the RAM stick. Measured in GBP(£).	More than 0, To two decimal places	19.99

Database Filename: DBparts.db

Table Name: TBLgpu

Description: This table will be used to store information about the graphics cards that are available.

Field Name	Data Type	Description	Valid requirements	example
GPU ID Primary key	String (can be converted to integer if required)	This is a unique number allocated to each Graphics card (record) in the table. This is so each record can be identified when retrieving information about the selected part.	More than 0	"0001"
Name	String	This is the name of the Graphics card	Not null	"Msi GTX 1050 ti Gaming X 4G"
Memory size	Integer	This stores the amount of video memory in each graphics card. This is so the user can sort by size and choose their preference. Measured in gigabytes (GB).	More than 0	4
CUDA cores	integer	This stores the amount of cores he GPU has. Therefore, the user can sort by the amount of cores and choose their preference.	More than 0	288
Memory type	String	This stores the type of memory the graphics card uses. This is so the user can filter by memory type and choose their preference.	Not null	"GDDR5"
GPU length	Float	This stores the length of the card to ensure that it fits into the case. Measured in centimetres (cm).	More than 0	17.0
GPU clock speed	Float	This stores the clock speed of in each graphics card. so the user can sort by speed and choose their preference Measured in gigahertz (Ghz).	More than 0	1.8
Power demand	Float	This stores the amount of power required to run the GPU so it can be added to the power budget to ensure the power requirements to run all the parts is met. Measured in watts.	More than 0	150

Price	Float	This stores the price of the Graphics card. Measured in GBP (£).	More than 0, To two decimal places	179.50
-------	-------	--	---------------------------------------	--------

Database Filename: DBparts.db

Table Name: TBLstorage

Description: This table will be used to store information about the secondary storage devices that can be added to the build.

Field Name	Data Type	Description	Valid requirements	example
Storage ID Primary key	String (can be converted to integer if required)	This is a unique number allocated to each secondary storage device (record) in the table. This is so each record can be identified when retrieving information about the selected part.	More than 0	"0001"
Name	String	This is the name of the storage device	Not null	"Samsung MZ-V6E500"
Storage port	string	This stores the port type that can be used to attach a secondary storage device to the motherboard.	Not null	"SATA"
Storage size	Integer	This stores the amount of memory in each storage drive. This is so the user can sort by size and choose their preference Measured in gigabytes (GB).	More than 0	128
Storage type	String	This stores the type of storage drive it is to allow the user to filter and choose their preference in storage type.	Not null	"SSD"
Power demand	Float	This stores the amount of power required to use the storage, so it can be added to the power budget to ensure the power requirements to run all the parts is met. Measured in watts.	More than 0	18.0
Price	Float	This stores the price of the storage drive. Measured in GBP (£).	More than 0 To two decimal places	19.99

Database Filename: DBparts.db

Table Name: TBLcase

Description: This table will be used to store information about the cases used for the builds.

Field Name	Data Type	Description	Valid requirements	example
Case ID Primary key	String	This is a unique number allocated to each secondary storage device	More than 0	"0001"

	(can be converted to integer if required)	(record) in the table. This is so each record can be identified when retrieving information about the selected part.		
Name	String	This is the name of the case	Not null	"NZXT S340 (White)"
Height	Float	This stores the height of the case to ensure the selected cooler fits within the case. Measured in centimetres (cm).	More than 0	9.75
GPU length	Float	This stores the maximum length of the graphics card that is compatible with the case. This is to ensure that it fits into the case. Measured in centimetres (cm).	More than 0	25.0
Form Factor	String	This stores the form factor of the motherboard that can fit inside the case; this is so it can be checked for compatibility with the selected case.	Not null	"Mini-ATX"
Price	Float	This stores the price of the storage drive. Measured in GBP (£).	More than 0 To two decimal places	59.99

Validation

For the tables above will be read only (cannot be changed) and will already be in a valid format. This means they require no additional validation is required for the data in the tables.

Saved builds Table

In order to save PC builds, I plan to make a procedure that creates a new table and recovers the primary keys of the selected parts to store them in the new table. This is so they can be recovered afterwards by researching the primary keys of the parts that need to be reloaded. Each record will be a saved build and will contain a name and primary key for each build.

Database Filename: DBparts.db

Table Name: TBLsavedbuild

Description: This table will be used to store the primary keys (as foreign keys) of the parts involved the PC build that is being saved.

Field Name	Data Type	Description	Valid requirements	example
Build ID Primary key	String (can be converted to integer if required)	This is a unique number allocated to each PC build (record) in the table. This is so each record can be identified when retrieving the parts in the build.	More than 0	"0001"
Name	String	This is the name of the build given by the user so they can recognise it on the load page.	Not null, Maximum length of 25	"Budget build 1"
Case ID Foreign key	String (can be converted to integer if required)	This is used to find the case of the build the user saved from the TBLcase table so it can be recovered	Is an actual primary key in TBLcase	"0001"

MOBO ID Foreign key	String (can be converted to integer if required)	This is used to find the motherboard of the build the user saved from the TBLmotherboard table so it can be recovered	Is an actual primary key in TBLmotherboard	"0001"
PSU ID Foreign key	String (can be converted to integer if required)	This is used to find the power supply unit of the build the user saved from the TBLpsu table so it can be recovered	Is an actual primary key in TBLpsu	"0001"
GPU ID Foreign key	String (can be converted to integer if required)	This is used to find the graphics card of the build the user saved from the TBLgpu table so it can be recovered	Is an actual primary key in TBLgpu	"0001"
CPU ID Foreign key	String (can be converted to integer if required)	This is used to find the CPU of the build the user saved from the TBLcpu table so it can be recovered	Is an actual primary key in TBLcpu	"0001"
CPUcooler ID Foreign key	String (can be converted to integer if required)	This is used to find the CPU cooler of the build the user saved from the TBLcpucooler table so it can be recovered	Is an actual primary key in TBLcpucooler	"0001"
RAM ID (1-4) Foreign keys	String (can be converted to integer if required)	This is used to find the RAM sticks of the build the user saved from the TBLram table so it can be recovered. There will be four of this field to match the maximum amount of ram sticks to put onto a motherboard. This is so if the user can select different types of ram for each slot.	Is an actual primary key in TBLram	"0001"
Storage ID (1-4) Foreign keys	String (can be converted to integer if required)	This is used to find the storage devices used in the build the user saved from the TBLstorage table so it can be recovered. There will be four of this field to match the maximum amount of storage devices to put onto a motherboard. This is so if the user can select different storage devices for each slot.	Is an actual primary key in TBLram	"0001"
Build Price	Float	This is used to store the price of the build so it can be displayed when the user loads the build. Measured in GBP(£).	Must be the sum of all the part prices in the build	£467.86
Power Usage	Float	This is used to store the power usage of the build so it can be displayed when the user loads the build. Measured in watts.	Must be the sum of all the power used by the parts in the build	252.4

Validation

the TBLsavedbuild will have data added to it. The data added to it that will go in the ID fields should be integers in

their respective parts tables. This will require a presence check to see if there is a record in ID being reference in the other tables. There will also need to be a range check for the Build price and Power usage to ensure that both values are more than one. This is because it would provide false information if the price and power usage of PC parts were negative.

Relationship



The relationship between all the entities the PC parts tables can be treated as one table. By treating it this way, it become easier to comprehend the relationship between those tables and the TBLsavedbuild table as a one to my relationship.

Internal Data structures

For now, I have a vague idea of what my app will be like. I have made this data structure table to document the data structure I am considering using in the pseudocode algorithms.

Dynamic data structures

These are the major data structures that change as result of a user interaction so require validation to ensure the users actions are understood and can be used affectedly by the program.

Variable/data structure name	Data Type/ structure	Scope	Description	Validation	Justification
ButtonIndex	Integer	Global	This variable stores the number allocated to the button the user has clicked to select a part. The number corresponds to the ID of the part the user has selected to load.	Range check	The ButtonIndex must be an ID in the databases table that is being searched. To ensure this is case the value will be compared to the highest numerical value in the ID column to find if it is less than or equal to the highest ID. This will make sure there is a record for the database to return.
CurrentPcBuildArray	String List	Global	This array stores the ID's of the parts involved in the current PC build so it can be converted into a record to be saved in the database. It is global so parts can be added and removed from any part of the code and it will remain consistent.	Length Check	A length check is need for the first index of the array, which stores the name of the PC given by the user. This is to make sure the field is not left empty when the PC is stored in the database and to ensure that the name is of a practical

					length for it to be displayed.
PcInfo	String	Local	This is used to store the string that displays the power usage, price and compatibility of the user's current PC build.	Presence check	A presence check is required to make sure that the string has the necessary data such as the available power output of the power unit. This is so it does not display false information about the PC build.
QueryResult	string	Global	This variables stores the values returned from the SQL query so it can be further manipulated and utilised for other procedures.	Presence Check	A presence check is required ins case the SQL query returns no values. By checking if the QueryResult is empty the program can notify the user that that data was recovered and stop further procedures using null values to display the result of the query.
SqlQuery	string	Global	This stores the SQL statement that will be used to query the database and retrieve or manipulate information from the databases. This is local because SQL Queries can be called from multiple parts of the code.	Format Check	A format check is need here to ensure that the SQLquery that has been created can be understood by the database to retrieve the requested information or add new information without causing errors.
SearchSqlQuery	string	Local	This variable is used to temporally store the search query in SQL form so that more parameters and be concatenated to the end of the string to add more search perimeters.	Format Check	A format check is need here to ensure that the SQLquery that has been created can be understood by the database to retrieve the requested information.
CompatibilitySocketsArray	String Array	Global	This is used to store the sockets of the parts that need to be checked for compatibility. Each index is aligned with the opposing index in CompatibilityPlugsArray so that each index can be compared to each other. If they are different the compatibility warning is raised.	Length Check	The validation for this array should be a length check to make sure it has the same amount of items as CompatibilityPlugsArray. This is so every item can be compared to the item in the same index in the CompatibilityPlugsArray array.
CompatibilityPlugsArray	String Array	Global	This is used to store the plugs of the parts that need to be checked for compatibility. Each index is	Length Check	The validation for this array should be a length check to make sure it has the same amount of

			aligned with the opposing index in CompatibilitySocketsArray so that each index can be compared to each other. If they are different the compatibility warning is raised.		items as CompatibilitySocketsArray. This is so every item can be compared to the item in the same index in the CompatibilitySocketsArray array.
ActiveSliderArray	Slider List	Global	This array stores the sliders that need to be activated to allow the user to input the filter perimeters that are needed to find the parts the user wants from the database.	Presence check	A presence check is needed to make sure that there are sliders in this array to activate and allow the user to input the data they want.
ActiveDropDownArray	DropDown List	Global	This array stores the sliders that need to be activated to allow the user to input the filter perimeters that are needed to find the parts the user wants from the database.	Presence check	A presence check is needed to make sure that there are Dropdown menus in this array to activate and allow the user to input the data they want.
Compatibility	Boolean	Local	This is Boolean value that will be change to true when parts are compatible and false the parts selected are not compatible.	None needed	This value is the result of the compatibility check, which can only give a result of true or false. both of which are valid values that this variable can store.

Constant data structures

These are data structures that will not change during the runtime of the program. This means they do not require any specific validation to ensure the data the constants contains is usable because they will be hardcoded to already valid values.

Variable/data structure name	Data Type/ structure	Scope	Description
HomeTextBoxArray	Textbox List	Local	This array stores the textboxes that are used to display information in the database as a data object. I have put them in an array so their index in the array can corresponds to the ID of the PC build they are displaying. It also allows me to cycle through them when populating them with data by incrementing the index of the textbox being filled with information.
FilterSlidersArray	Slider List	Local	This stores all the sliders that can be used to input filter parameters. I put the sliders in an array so that it can be cycled thought to set each slider to active or deactivated to make sure certain sliders are on active when they have no effect on the search query.
FilterDropDownArray	DropDown List	Local	This stores all the dropdown menus that can be used to input filter parameters. I put the dropdown menus in an array so that it can be cycled thought to set each dropdown menu to active or deactivated to make sure certain sliders are on active when they have no effect on the search query.

PcTemplateArray	String Array	Global	The is array is used to store the location of each type of part in the CurrentPcBuildArray so when a part is added to the build the index that it should be placed in can be can be found.
PartSpriteArray	Sprite Array	Local	This array stores the sprites used to display the parts in the build to the user. This is so they can be cycled through and activated if a part in their index from the CurrentPcBuildArray to show that parts have been selected.

Algorithms

The program will be program will be comprised of algorithms that fit together as a series of modules based off the decomposition diagram I created earlier. I have created some pseudocode to explain the algorithms I am considering using. The heading of each algorithm is the name I am planning to use for the procedure when I implement it into the program during development.

CameraMovement

```

01 WHEN page button is clicked
02 Set Camera X and Y co-ordinate to the predefined position depending on the button and selected page.
03 Set quit button X and Y co-ordinate to the predefined position depending on the button and selected page.
04 Set Build page button X and Y co-ordinate to the predefined position depending on the button and selected page.
05 Set Home page button X and Y co-ordinate to the predefined position depending on the button and selected
...page.
06 Set Selection page Button and Y co-ordinate to the predefined position depending on the button and selected
...page.

```

This pseudocode simply describes how each button that moves with the camera will have their location on the editor change so they appear where the camera is displaying parts of the app. I can see this being implemented as a procedure, where the coordinates of each button and the camera is passed into the procedure as parameters. Each button will pass the appropriate set of coordinates by value into the procedure.

LoadPc

```

01 WHEN a LoadPc button is clicked
02 INPUT ButtonIndex
03 OPEN DBparts.db Table TBLsavedbuild
04 Create SQL Query to select record where ButtonIndex = buildID
05 currentPcArray = query result
06 CALL DisplayCurrentBuild procedure

```

This pseudocode describes how saved PCs can be recovered from the database. This satisfies an aspect of success criteria and the [2.1.X.X] module from the decomposition table. Lines 01 and 02 take the users selection by first initiating this algorithm from the user clicking a button and then finding index in the button to use as an identifier to find the PC the user request from the database. Line 03 opens the TBLsavedbuild table to retrieve the data of the requested PC and line 04 is the SQL query that is generated to find the PC the user wanted, using the ButtonIndex variable to identify the primary key of the requested PC data. In Lines 05 and 06 the data from the SQL query is stored as the currentPcArray array and DisplayCurrentBuild procedure is called to refresh the display to present the recovered PC.

SavePc

```

01 WHEN save button is clicked
02 OPEN DBparts.db Table TBLsavedbuild
03 IF Buildname not null:
04     THEN IF Buildname in TBLsavedbuild names field:
05         THEN replace PC with buildname as name

```



```

06         ELSE create SQL for a new record in TBLsavedbuild using CurrentPcBuildArray
07         CALL UpdateSavedBuild procedure
08     ELSE OUTPUT "Invalid name" message

```

This pseudocode describes how PCs will be stored when a save button is clicked. This satisfies an aspect of success criteria and the [2.2.X.X] module from the decomposition table. Lines 01 and 02 initiate this algorithm and open the TBLavedbuild table. Line 3 checks to see if the user has entered a name for the PC build as part of the validation discussed earlier. If a name is given, the algorithm checks if there is already a PC build with the given name(line 04). If there is, the data of the record with the same name is replaced with the new PC build (line 05). If the name does not exist in the table, a new record is made to store the current build (line 06). The UpdateSavedBuild is then called to make sure the saved PC builds list is until up to date. If a PC name is not given the program will display that a message to tell the user to name the PC (line 07).

DeletePc

```

01 WHEN a remove button is clicked
02 INPUT ButtonIndex
03 OPEN DBparts.db Table TBLsavedbuild
04 Create SQL Query to Delete record where button index = buildID
05 Shift all buildIDs to their current value - 1 where their BuildID > buttonIndex
06 CALL UpdateSavedBuild procedure

```

This pseudocode describes the process of how saved PCs will be deleted. Lines 01 and 02 (like the LoadPc procedure) take the users selection by first initiating this algorithm from the user clicking a button and then finding index in the button to use as an identifier to find the PC the user wants to remove from the database. Line 03 opens the TBLsavedbuild table and 04 runs the SQL instructions to remove the PC using the ButtonIndex to identify the PC the user wants to remove. In line 05 all the records with an ID more that the record that was deleted, change their ID to one less that its current value. This is to insure that the high value ID represents the amount of records in the table. Finally line 06 refreshes the list of saved PC builds.

UpdateSavedBuild

```

01 OPEN DBparts.db Table TBLsavedbuild
02 Create SQL Query to select the first record
03 FOREACH Textbox in HomeTextBoxArray:
04     DO Textbox.Text = the Name attribute of then Query result
05     IF the next record is not null
06         THEN Create SQL Query to select the next record
07     ELSE Break loop

```

This pseudocode describes how the home page will be filled with the names of the saved PC build so the user can see which build they want to load or delete. Line 01 opens the TBLsavedbuild table and line 02 selects the first record in the table. Then line 3 starts a loop that cycles through the textboxes in the homepage stored in HomeTextBoxArray Array. Lines 04 to 08 fill the textboxes with the name attribute of the PC in the TBLdavedbuild table. Then the program checks if there are any more records in the table, if there are, the next record is selected and the loop repeats. If there are no more records the loop is ended.

UpdateFilters

```

01 After PartTypeDropDown selection
02 POPULATE ActiveSliderArray and ActiveDropdownArray with the siders and dropdowns that should be active
03 FOREACH Slider in FilterSlidersArray:
04     slider. Deactivate

```

```

05 FOREACH Dropdown array:
06     Dropdown.Deactivate

07 FOREACH Slider in tempSliderarray:
08     slider.activate
09 FOREACH Dropdown in tempDropdoenarray:
10     Dropdown.activate

```

This pseudocode describes how certain dropdown menus and sliders will appear on the selection page depending on the part type the user selected. This is needed to allow the user to filter the PC parts to find what they want. This is to fulfil a success criteria and is a key part of the [3.1.X.X] modules from the decomposition tree. Line 1 starts this procedure once a selection has been made from the PartTypeDropDown dropdown menu. In line 02 the sliders that need to be activated are put into the ActiveSliderArray array and the dropdown menus that need to be activated are put into the ActiveDropdownArray array. In lines 03 and 04 all the sliders are deactivated so that useless sliders do not appear on the screen and confuse the user. Lines 05 and 06 do the same but for the dropdown menus in FilterDropdownArray array. Finally, lines 07 to 10 activate the sliders and dropdown menus in the ActiveSliderArray and ActiveDropdownArray arrays.

PartSearch

```

01 OPEN DBparts.db Table ("TBL"+PartTypeDropDown.value)
02 QueryCondition = (QueryCondition + "Price <=" + PriceSlider.value)
03 IF search bar not null:
04 THEN QueryCondition = (QueryCondition + " AND Name == " + Searchbar.Text )
05 FOREACH Slider in SliderArray:
06     IF Slider.enabled = TRUE:
07         THEN QueryCondition = (QueryCondition + " AND " + slider.name + " <=" "slider.value")

08 FOREACH Dropdown in DropdownArray:
09     IF dropdown.enabled = TRUE:
10         THEN IF Dropdown.value != Dropdown.name
11             THEN QueryCondition = (QueryCondition + " AND " + "Dropdown.name" + " = "
...+"Dropdown.value")

12 Create SQL Query with QueryCondition
13 FOREACH Textbox in SelectionTextBoxArray:
14     DO Textbox.Text = the Name and attributes of the Query result
15     IF the next record is not null
16         THEN Create SQL Query to select the next record
17         ELSE Break loop

```

This pseudocode describes how the filter parameters chosen by the user are converted into a database query that can find the parts that fit the parameters. This is an important feature that aids the user experience by eliminates the PC parts they are not interested in. The algorithm starts by opening the table containing the part type the user has selected from the dropdown menu. Then a new search query is made that uses the value of the price slider as the first condition. The condition ensures that the PC parts returned from the database have a price that is less than the value selected by the user. Lines 03 and 04 check if the user has input a keyword for the part they are looking for. If they have, a condition is concatenated to the search query where the keyword is in the name of the PC part. Lines 05 to 07 check to see which sliders are enabled. If they are enabled, the name of the slider is used to identify the name of the field it applies to and the sliders value is used to set as the condition a record also has to meet in order to me returned from the database. In lines 08 to 11 the dropdown menu goes through the same process as the siders to see if they are enabled. If they are, they are checked to make sure the name of the dropdown menu is not the current value selected from the menu, which would mean that the user has not selected any conditions form

the dropdown menu. If the dropdown menu's name is not the value selected from it, then the name and value of the dropdown menu is used as a condition in the search query. In line 12 the search query that was made from the search bar, sliders and dropdown menus is run. In line 13 to 17 of the values returned from the query is used to fill textboxes the information of each recovered that fulfils the requirements the user has given.

ResetFilters

```
01 WHEN reset button is clicked
02 Searchbar.text = null
03 SET all slider values to max value
04 SET all Dropdown to option 1
05 CALL PartSearch Procedure
```

This pseudocode sets all the filtering tools to a default value that will display all the parts in the table depending of the parts table selected from the PartTypeDropdown dropdown menu. This is a more convenient option for the user as they do not have to manually reset each filter option. Line 01 begins this procedure from the reset button being clicked. Line 02 sets the search bar's text to null so that the query avoids using it for the search. Without this, errors may arise where the program cannot search for a null value. Line 03 sets the slider values to the maximum so that all values are allowed, resulting in all the parts being displayed. Line 04 sets all the dropdown menus to option one, which should allow for any value to be accepted, also resulting in all PC parts being displayed. Finally line 05 calls the PartSearch procedure so that all the parts can be displayed.

AddPart

```
01 WHEN a LoadPart button is clicked
02 INPUT ButtonIndex
03 CurrentPcBuildArray [index of PartTypeDropdown.value in PcTemplateArray] = ButtonIndex
04 OPEN DBparts.db Table ("TBL"+PartTypeDropdown.value)
05 Create SQL Query to select the Build Price and power usage where Button Index = Build ID
06 CurrentPcBuildArray [Price Index] += build price from Query result
07 CurrentPcBuildArray [Power Index] += Power usage from Query result
08 CALL DisplayCurrentBuild procedure
```

This pseudocode describes how a new PC parts will be added to the build. This needs to happen order to allow the user to build a PC and fulfil the main purpose of the app. After a load button is clicked and the button index is stored from lines 01 and 02. The location the part should take in the CurrentPcBuildArray is found by finding the location (index) of the type of part in the PcTemplateArray given by the PartTypeDropdown menu's value. Once the location of the part is found, the ButtonIndex is stored in the location in the CurrentPcBuildArray array to use as reference to the part. In lines 04 and 05, the price and power usage of the part is taken from the table the new part's information is stored in. The price and power usage is added to that of the current build in lines 06 and 07. Finally, the DisplayCurrentBuild procedure is called to update the information displayed about the PC because it would have changed as a result of the newly added part.

RemovePart

```
01 WHEN a RemovePart button is clicked
02 INPUT ButtonIndex
03 OPEN DBparts.db Table ("TBL"+PcBuidTemplate[ButtonIndex])
04 Create SQL Query to select the Build Price and power usage where CurrentPcBuildArray [ButtonIndex] = Build ID
05 CurrentPcBuildArray [Price Index] -= build price from Query result
06 CurrentPcBuildArray [Power Index]- = Power usage from Query result
```

07 CurrentPcBuildArray [ButtonIndex] = Null
08 CALL DisplayCurrentBuild procedure

This pseudocode describes how parts will be removed from the PC build this is part of the success criteria and module [1.1.2.0] of the decomposition table. It begins when a remove button I clicked. In lines 02 and 04, the index of the remove button that was clicked is then used to find the part table that needs to be opened to get the price and power usage of the part being removed. In lines 05 to 07 the price and power usage of the part is subtracted from the PC build's price and power usage. In line 07 the PC part in the CurrentPcBuildArray set to null so when the DisplayCurrentBuild procedure is called, the part will not be displayed.

UpdatePcInfo

```
01 PcInfo = ("Build Price: " + CurrentPcBuildArray [Price Index] + "\nPower Usage: " + CurrentPcBuildArray [Power
...Index] + "/")
02 OPEN DBparts.db Table TBLpsu
03 Create SQL Query select power rating where CurrentPcBuildArray [Power Index] = Build ID
04 PcInfo = (PcInfo + Power usage from Query result)
05 OUTPUT PcInfo

06 FOREACH socket in CompatibilitySocketsArray:
07     DO IF socket == null or CompatibilityPlugsArray [capability index] == null
08         THEN pass
09         ELSE IF socket != CompatibilityPlugsArray [capability index]:
10             THEN Compatibility = False

11 IF compatibility == False:
12 THEN OUTPUT "some of the parts you have selected are not compatible"
```

This pseudocode describes how the information about the PC will be updated to ensure it is accurate to the current build as to not miss inform the user. This algorithm starts by creating a concatenated string that takes the price and power usage of the build from the CurrentPcBuildArray array. Then in lines 02 and 03, the TBLpsu is opened open to get the power rating of the power supply used in the current build. This is then added to the end of the concatenated string in line 04 which is then outputted.

Line 06 to 10 starts a loop that cycles though each index of the CompatibilitySocketsArray and checks if it matches its corresponding index in the CompatibilityPlugsArray given it is not a null value. If one of the of the values is null, the program passes to the next index and if they do not match then the Compatibility variable is set to false so that the program can display that the parts are not compatible in lines 11 and 12.

DisplayCurrentBuild

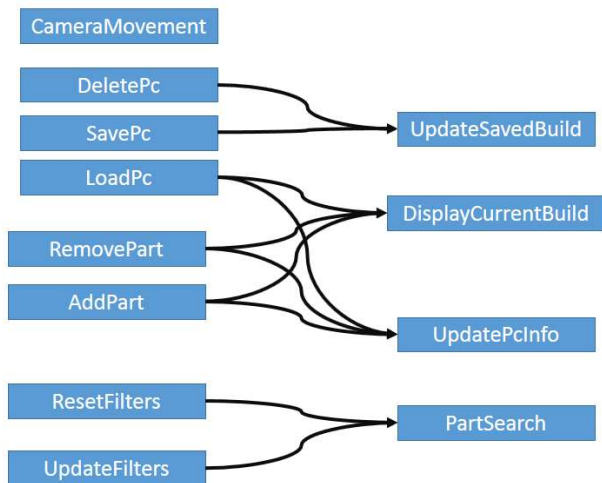
```
01 FOREACH Part in CurrentPcBuildArray:
02     DO IF Part == null
03         THEN hide PartSpriteArray[PartsIndex]
04         ELSE Display PartSpriteArray[PartsIndex]
05     PartsIndex+=1
06 Call UpdatePcInfo
```

This pseudocode describes how the parts in the current build will be displayed for the user to see which parts have been selected. Line 01 starts a loop that cycles through each part in the CurrentPcBuildArray array. In lines 02 to 04 check the current index in the CurrentPcBuildArray and if the value is not null the sprite in the PartSpriteArray with the same index will be displayed inform the user of that part been selected. If the value is null, the corresponding

sprite will be hidden to inform the user that the part has not been selected as a form of validation. Finally line 05 increments the PartsIndex variable in order make sure the corresponding sprite is displayed.

Line 6 calls the UpdatePcInfo procedure to make sure the information about the PC is accurate.

Algorithm links



This diagram shows the links between the algorithms. These links are where a procedure calls another procedure. By linking them I am able avoid repeating code because the same lines of code are accessible by each procedure.

The CameraMovement procedure is not linked to any other function because it does not require any repeated code.

The DeletePc and SavePc build procedures both all the UpdateSavedBuild procedure because they both affect the data in the TBSavedBuild table, so the changes they make will need to be displayed.

The LoadPc, RemovePart and AddPart procedures all affect the PC parts involved in the user's current build; as a result, the DisplayCurrentBuild and UpdateInfo procedures need to be called to ensure the interface is displaying the correct information. The ResetFilters and UpdateFilters both need to retrieve and display parts from the database on the part selection page. Because of this, they both call the PartSearch procedure to show the parts available for the user to select.

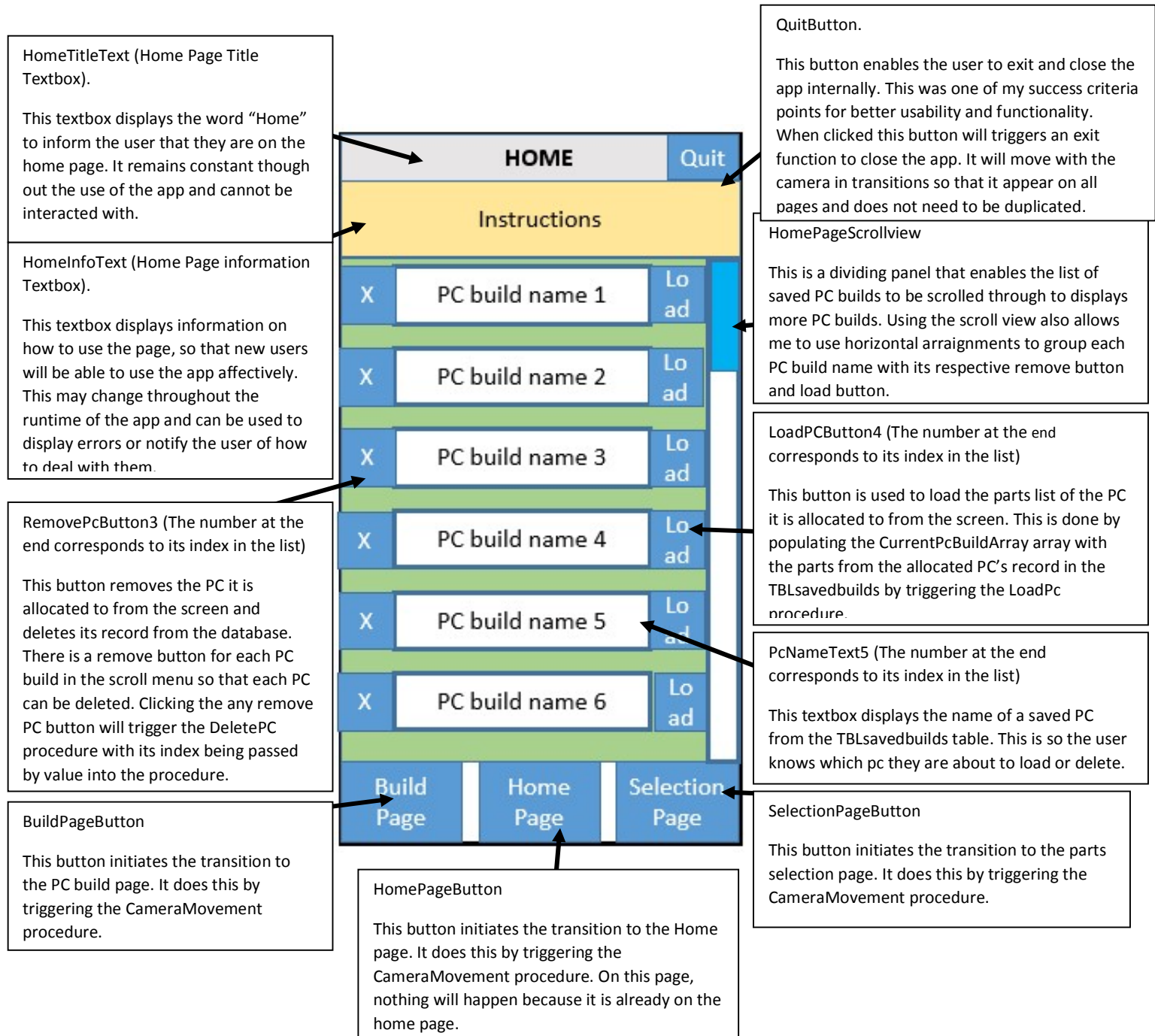
User interface

The graphical user interface (GUI) is the main method to interact with mobile applications. For my app, I plan to have three pages for saving and loading PC builds, selecting parts and viewing the PC. My stakeholders at Webx have advised me to render all the screens at the same time and use the camera (view area) in the editor to zoom into the page that is currently being used. When the page needs to change only the camera and repeated interface buttons will move to the position of the requested page. This gets rid of the rendering time that would occur if all the pages were separate. For a better user experience and to avoid repeating code. I have decided to that the quit and page navigation buttons will move with the camera, so they are available on all the pages to enable the user to go to any of the pages on one click and to exit the app. Webx also advised me to use a portrait screen orientation for my app, as it is the most comfortable way for the user to interact with their device and therefore the app, which helps with the user experience. I will also use an aspect ratio of 16x9 because it is the most common aspect ratio for modern mobile devices and can easily be rescaled to the fit screen size as required without too much distortion; this makes the app compatible with a wider variety of devices. I have used the camel cap naming convention to name the parts, so that I can recognise the function of each part and the kind of UI element it is from the last word. For example,

“SaveBuildButton” is the button used to save builds. The Capital letters indicate the start of new words so there are no spaces in the name in order for me to reference within in the scripts.

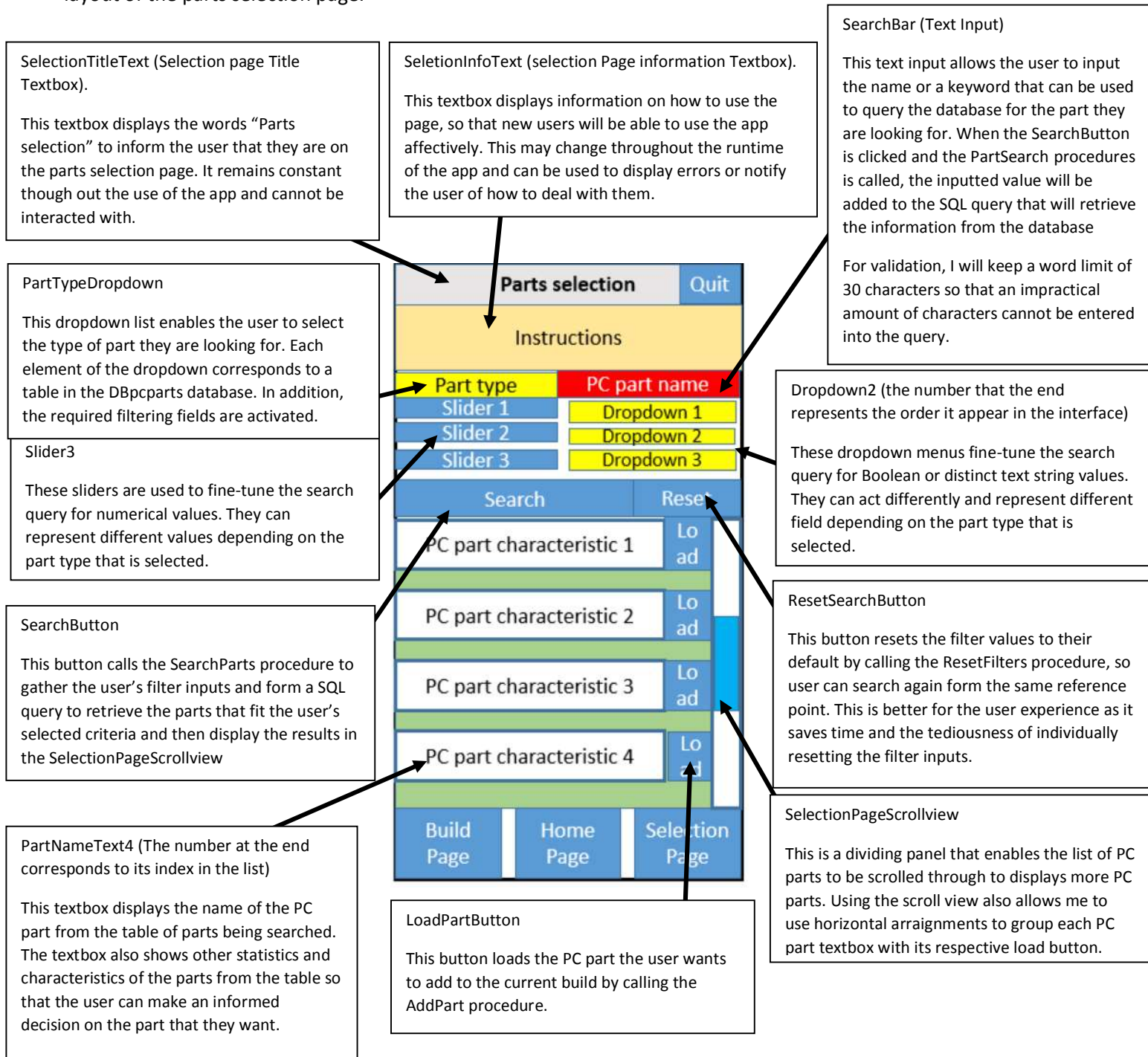
Home page (Load PC page)

The first page will be the page where they can manage their saved PC builds; I am using this as the home page because a frequent user of the app will want to the load PC builds they are working on and this gets them to the page they need as soon as the app has started by default. This gives a better user experience. The diagram below shows the layout of the Load PC page.



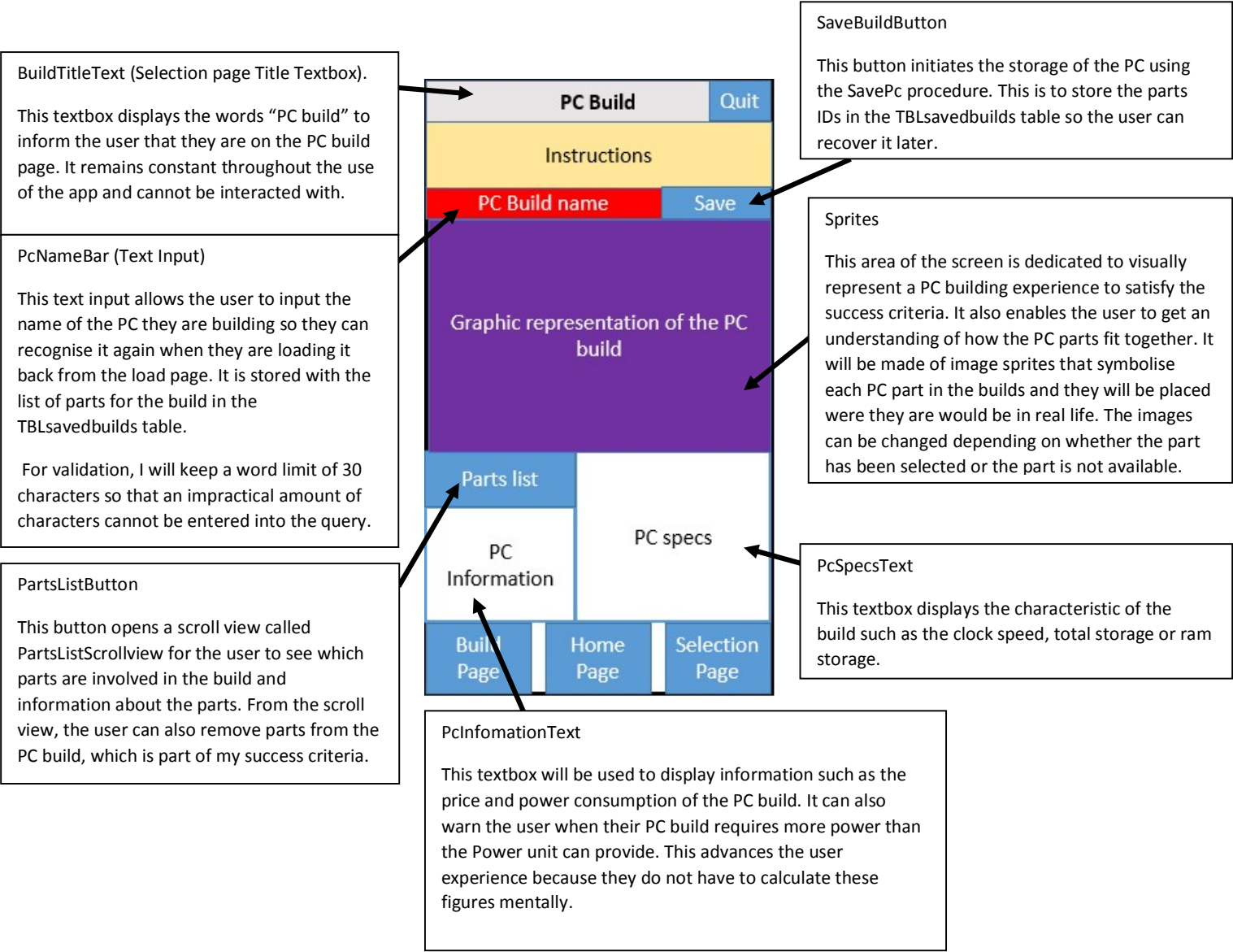
Parts section page

This page is where the user can find the parts that they want from the database. They can adjust their query using a filtering menu of sliders, Dropdown and a keyword search bar. A report will then be displayed underneath of the parts that were found. Then the user can load the part that they want into their build. The 'QuitButton', 'BuildPageButton' and 'HomePageButton' are the same as in the Home page and function in the same way. The 'SelectionPageButton' does nothing because the user is already on the selection page. The diagram below shows the layout of the parts selection page.



PC build page

This is the page where the user can inspect their PC build and the parts involved in the build. This page will allow the user to name their build and see how the PC parts fit together to satisfy the success criteria of “The application must visually represent a PC building experience”. It will also display the specifications and other information about the PC so they can make a better-informed decision on the changes they may need to make or whether there is a n issue with their PC build. The ‘QuitButton’, ‘SelectionPageButton’ and ‘HomePageButton’ are the same as in the Home page and function in the same way. The ‘BuildPageButton’ does nothing because the user is already on the selection page. The diagram below shows the layout of the Build page.



Implementation strategy

As discussed in my analysis section, I am using an agile development management system for this build. This is an iterative development system, which entails analysis (requirements), design, implementation, testing and evaluation stages that repeat and advance the product by taking new testing and evaluation feedback to make new requirements that are implemented on the next cycle. So far, I have made and initial analysis and design stages of

the cycle. As I now transition into the first implementation phase, I have made a plan of the tasks I would like to complete on each iteration. In each iteration, a prototype will be made that for testing by my stakeholders to provide feedback on the changes that I need to make. I will then design and implement these new changes along with the planned tasks to implement for the next iteration. I have four pre-planned iterations I would like to implement and test before implementing additional iterations. Additional iterations can and can be added if there are changes to be made to the prototype after the four planned iterations have been completed.

Using the decomposition table as a guide here are the iterations module I have planned:

Iteration 1: User Interface

This is the first iteration and it involves the creation and functionality of the interface. The resulting prototype will give the stakeholders and testers a view of how the interface fits together. It also will give them a basis to provide feedback on the usability of the app before the other functions and procedures are added to the app.

This iteration consists of:

- Import all the elements that will comprise the user interface (i.e. the textboxes, sprites, sliders, etc.).
- Page transitions (the CameraMovement procedure.)
- Temporary button functionality (actions that the interface elements can do in order to show the testers (user) what each element will do even if its proper function has not been implemented yet. E.g. a prompt to tell the user that a PC part has been loaded.)

Iteration 2: Adding and Removing Parts

This is the second iteration and involves implementing the procedures and functions that are involved in the adding and removing of parts from the PC build. This will be the implementation of the decomposition modules [1.1.X.X], [1.2.X.X] and [4.0.0.0]. The prototype created from the implementation of this iteration will be used to test the process of adding and removing of parts from the PC build. This iteration will also give me the opportunity to implement any required changes to resolve problems the user testing found after the first iteration.

This iteration consists of:

- Implement any changes required from feedback from the first iteration
- Create the database with the part tables
- Display information about the PC (UpdatePcInfo procedure)
- Display the parts in current build (DisplayCurrentBuild procedure)
- Add parts from the build (AddPart Procedure)
- Remove Parts from the build (RemovePart procedure)

Iteration 3: Saving, Loading and Deleting of PC Builds

This is the third iteration and involves implementing the procedures and functions that are involved in the saving, loading and deleting of PC builds from the database. This will be the implementation of the decomposition modules [2.1.X.X] and [2.2.X.X]. The prototype created from the implementation of this iteration will be used to test the management of saved builds. This iteration will also give me the opportunity to implement any required changes to resolve problems the user testing found after the other iterations.

This iteration consists of:

- Implement any changes required from the prior iterations
- Create a TBLsavedBuild table
- Save PC builds (SavePc procedure)
- Delete previously saved PC builds (DeletePc procedure)
- Update the list of saved PC builds (UpdateSavedBuild procedure)

Iteration 4: Parts Filtering System

This is the fourth iteration and involves implementing the procedures and functions that are involved in the filtering of parts in the selection page. This will be the implementation of the decomposition module [3.1.X.X]. The prototype created from the implementation of this iteration will be used to test the functionality and effectiveness of the filtering system. This iteration will also give me the opportunity to implement any required changes to resolve problems the user testing found after the other iterations.

This iteration consist of:

- Implement any changes required from the prior iterations
- Reset filter values (ResetFilters procedure)
- Change the available filters depending on the part type selected (UpdateFilters procedure)
- Recover and display parts from the database (PartSearch procedure)

Additional Iterations

If there are until changes to be made after, the four iterations I will add more iterations and develop a new plan to make the required changes and the testing required for the new prototypes. This will happen until a prototype is deemed a suitable and functioning solution. At which point the final prototype will go under final testing and an evaluation to assess its success and possible further development possibilities.

Testing strategy

I will need to test the features that I implement to ensure that the program functions correctly, as I develop it. When developing I plan to use white box testing, in which I can inspect the internal mechanics of the program for problems as the program processes the user's inputs. White box testing will enable me to find logical errors and optimize section of code during the debugging of the implemented code.

At the end of each iteration, before giving the prototype to the stakeholders to provide feedback, I will use black box testing to assure the functional integrity of the prototype and ensure that key aspects of the program that are required to fit the success criteria work correctly. These black box tests include:

- Inputting valid, borderline and invalid data into text fields to ensure the all kinds of inputs are handled properly.
- Clicking all the buttons available to ensure they perform a useful function.
- Ensuring the correct PC parts are retrieved from the database.
- Ensuring the correct PC build is retrieved from the database.
- Ensuring the PC build information accurate

Testing during development

In each iteration, new features are implemented into the solution. The added features from each iteration will require testing to ensure they work. I have identified some testing that I will use for my planned iterations. More may be added for further iterations. Within each iteration, I will use white box testing to debug the code I implement and at the end of each iteration I will use black box testing to test the effectiveness of the prototype from the users perspective.

Iteration 1: User Interface Testing

Here are some tests I will use to ensure the features I will implement in the first iteration function correctly.

Test number	Test Data	Test/data type	What should happen	justification
-------------	-----------	----------------	--------------------	---------------

1	Clicking the Quit Button	Valid	When the Quit Button is clicked, the app closes.	To ensure that the program can be closed properly by the user as to not use up resources while running in the backgrounds of the users device.
2	Clicking the Selection Page Button	Valid	When the Selection Page Button is clicked the page that is displayed will change to the parts selection page.	This is to make sure the user can go to the Selection Page to add a new at any time. There is only a valid data type because there is no way to make an invalid input on a button.
3	Clicking the Home Page button	Valid	When the Home Page button is clicked the page that is displayed will change to the Home Page.	This is to make sure the user can go to the Home Page to load or delete a PC at any time. There is only a valid data type because there is no way to make an invalid input on a button.
4	Clicking the Build Page button	Valid	When the Build Page button is clicked the page that is displayed will change to the PC Build Page.	This is to make sure the user can go to the PC build page to inspect and make changes to there build. There is only a valid data type because there is no way to make an invalid input on a button.

Iteration 2: Adding and Removing Parts Testing

Here are some tests I will use to ensure the features I will implement in the second iteration function correctly. New tests may be added to test the changes made from feedback of the first iteration.

Test number	Test Data	Test/data type	What should happen	Justification
1	Select "Motherboard" from the parts dropdown menu	Valid	When selected the parts list should display all the available motherboards and information about these motherboards from the TBLmotherboard table.	This is to ensure that that TBLmotherboard table can be properly accessed and read by the program, in order to display a range of motherboards to the user, so they can choose which part they want.
2	Select "Case" from the parts dropdown menu	Valid	When selected the parts list should display all the available cases and information about these cases from the TBLcase table.	This is to ensure that that TBLcase table can be properly accessed and read by the program, in order to display a range of cases to the user, so they can choose which part they want.
3	Select "Power Supply" from the parts dropdown menu	Valid	When selected the parts list should display all the available power supplies and information about these power supplies from the TBLpsu table.	This is to ensure that that TBLpsu table can be properly accessed and read by the program, in order to display a range of power supplies to the user, so they can choose which part they want.
4	Select "CPU" from the parts dropdown menu	Valid	When selected the parts list should display all the available CPUs and information about these CPUs from the TBLcpu table.	This is to ensure that that TBLcpu table can be properly accessed and read by the program, in order to display a range of CPUs to the user, so they can choose which part they want.
5	Select "CPU Cooler" from the	Valid	When selected the parts list should display all the	This is to ensure that that TBLcpucoolers table can be properly

	parts dropdown menu		available CPU coolers and information about these coolers from the TBLcpucooler table.	accessed and read by the program, in order to display a range of coolers to the user, so they can choose which part they want.
6	Select “GPU” from the parts dropdown menu	Valid	When selected the parts list should display all the available GPUs and information about these GPUs from the TBLgpu table.	This is to ensure that that TBLgpu table can be properly accessed and read by the program, in order to display a range of GPUs to the user, so they can choose which part they want.
7	Select “RAM” from the parts dropdown menu	Valid	When selected the parts list should display all the available RAM sticks and information about these RAM sticks from the TBLram table.	This is to ensure that that TBLram table can be properly accessed and read by the program, in order to display a range of RAM sticks to the user, so they can choose which part they want.
8	Select “Storage” from the parts dropdown menu	Valid	When selected the parts list should display all the available storage devices and information about these storage devices from the TBLstorage table.	This is to ensure that that TBLstorage table can be properly accessed and read by the program, in order to display a range of storage devices to the user, so they can choose which part they want.
9	Add the first part in the parts list	Valid	When a displayed part in the list is selected to be added, the exact part should be added to the current PC build and displayed in the PC build page. The PC build information should also update to the specs of PC build with the part that was added.	This is to ensure that the part selected from the parts list by the user will be displayed on as a part of the PC build and the information about the PC build is changed to the appropriate information that accounts for the newly added part.
10	Add a part in the middle of the parts list	Valid	When a displayed part in the list is selected to be added, the exact part should be added to the current PC build and displayed in the PC build page. The PC build information should also update to the specs of PC build with the part that was added.	This is to ensure that the part selected from the parts list by the user will be displayed on as a part of the PC build and the information about the PC build is changed to the appropriate information that accounts for the newly added part. This test is needed to make sure that the program does not add the same index in the list. Additionally, this test makes sure the previously selected part that occupies position the new part in meant to be in, is removed and the new parts data is used in its place.
11	Add the last part in the parts list	Boundary (valid)	When a displayed part in the list is selected to be added, the exact part should be added to the current PC build and displayed in the PC build page. The PC build	This test ensure that no errors will be caused if the user selects the last available part in the parts list and part selected will be displayed on as a part of the PC build and the information about the PC build is changed to the

			information should also update to the specs of PC build with the part that was added.	appropriate information that accounts for the newly added part.
12	Add a part in a blank slot in the parts list	Invalid	When a blank slot in the parts list is selected to be added, nothing should happen because there is no part to add to the build.	This is to ensure that errors do not appear when the program tries to load a part without any data about which part it should load.
13	Remove a part from a blank slot in the PC build	Invalid	When a blank slot in the PC build is selected to be removed, nothing should happen because there is no part that is selected to be removed.	This is to ensure that errors do not appear when the program tries to remove part without any data about which part it should remove.
14	Remove a part from a filled slot in the PC build.	Valid	When a displayed part is selected to be removed, it should no longer be displayed and the PC build information should update to the specs of PC build without the part that was removed.	This is to ensure that the user will be able to remove the parts they do not want to be in the PC build and the information about the PC Build will be updated to an accurate value as a result of it.
15	Add a part that is incompatible with another already selected part. (i.e. selecting a motherboard with a DDR3 socket and RAM with a DDR4 socket)	Invalid	When the incompatible part is selected the compatibility warning is displayed as part of the PC information.	This is to ensure that the compatibility warning appears when the parts selected by the user are not compatible.

Iteration 3: Saving, Loading and Deleting of PC builds Testing

Here are some tests I will use to ensure the features I will implement in the third iteration function correctly. New tests may be added to test the changes made from feedback of the prior iterations.

Test number	Test Data	Test/data type	What should happen	justification
1	Select some parts and click the Save button – without adding a name	Invalid	When the saved button is clicked a prompt should appear to tell the user to add a name for the PC build.	This is to ensure that PCs are not saved without name, which would mean the user cannot recognise the PC the saved when trying to load it back.
2	Select some parts and click the Save button – with a valid PC name (e.g. “Budget PC Build”)	Valid	When the save button is clicked the current PC that is being displayed should be saved to the TBLSavedBuilds table and appear on the Home page.	This is to make sure that records in the form of PCs can be added to the TBLSavedBuilds table correctly and will appear on the saved build page.

3	Select some parts and click the Save button – with a PC name that is too long (e.g. “This PC has a really long and impractical name”)	Invalid	When the saved button is clicked a prompt should appear to tell the user the name they have used is too long.	This is to ensure that the word limit functions correctly, so the user cannot save a PC with a name that is too long to be displayed on the Home page.
4	Select some parts and click the Save button – with a PC name that is just above the limit (e.g. “This PC has a really long and impractical name”)	Boundary (Invalid)	When the save button is clicked a prompt should appear to tell the user the name they have used is too long.	This is to make sure the word limit works at correctly at the boundary of the name being too long. In this case the name is too long.
5	Select some parts and click the Save button – with a PC name that is just within the limit (e.g. “This PC has a long but practical name”)	Boundary (Valid)	When the save button is clicked the current PC that is being displayed should be saved to the TBLSavedBuilds table and appear on the Home page.	This is to make sure the word limit works at correctly at the boundary of the name being too long. In this case the name is short enough.
6	Click the save button with a PC same name of an already saved PC (e.g. “Budget PC Build”)	Valid	When the save button is clicked the current PC that is being displayed should replace the PC saved under the same name in the TBLSavedBuilds table.	This test is to ensure that records in the TBLSavedBuilds table can be updated to store the user’s changes to a previously saved PC build.
7	Load a PC in a blank slot in the saved Build list	Invalid	When a blank slot in the saved build is selected to be loaded, nothing should happen because there is no build to be loaded.	This is to make sure no errors are caused by the user tries to load a PC that does not exist in the list.
8	Load a PC in filled slot in the saved Build list	Valid	When the load button is clicked, the PC parts used in the build that was selected should be displayed on the PC build page.	This is to ensure that the TBLSavedBuilds table can be read and the parts I a previously saved PC build ca be recovered.
9	Delete a PC in a blank slot in the saved Build list	Invalid	When a blank slot in the saved build list is selected to be deleted, nothing should happen because there is no build to be deleted.	This is to make sure no errors are caused by the user tries to delete a PC that does not exist in the list.
10	Delete a PC in filled slot in the saved Build list	Valid	When the delete button is clicked the selected PC’s	This is to ensure that the receords in the TBLSavedBuilds table can be

			record is removed from the TBLSavedBuilds table.	removed in order to delete a saved PC.
--	--	--	--	--

Iteration 4: Parts Filtering System Testing

Here are some tests I will use to ensure the features I will implement in the third iteration function correctly. New tests may be added to test the changes made from feedback of the prior iterations.

Test number	Test Data	Test/data type	What should happen	justification
1	Input the name of a part in the list into the search bar and click the search button. (e.g. "NVidia GTX 1080 Ti")	Valid	When the search button is clicked only the part with the name given should appear on the parts list.	This is to make sure the user will be able to search for a specific part using its name and the program is able display the part in the list.
2	Input part of a name that is in the list into the search bar and click the search button. (e.g. "GTX")	Boundary (valid)	When the search button is clicked the parts with the searched text in their name will appear on the parts list.	This is to make sure the program can return a list of parts that contain the user input, in case the user cannot remember the specific name of the product they are looking for.
3	Input the name of part into the search bar that is not in the list and click the search button. (e.g. "SpaceBar!!!")	Invalid	When the search button is clicked no parts will appear because there are no parts that have the text given I there name.	This is to make sure no errors are made if the program cannot find the part the user was looking for.
4	Change the price slider to a lower value, but not 0 and click the search button. (e.g. 100.00)	Valid	When the search button is clicked only the parts with a price that is less than the value allocated by the price slider appear in the list.	This is to make sure the filtering system can get rid of parts in the list that have a price that is higher than the one the user has given.
5	Change the price slider to 0 and click the search button.	Invalid	When the search button is clicked no parts will appear on the list because none of the parts have prices of 0.	This is to make sure no errors are made if the program cannot find the part the user was looking for.
6	Change the values of the available sliders to filter out some options Note: this test will be done for all the types of parts, so data will change.	This test is dependent on the range of parts and siders available	When the search button is clicked only the parts that fit the criteria given or better are displayed.	This to make sure the appropriate parts the user wants are displayed when they input multiple perimeters using the sliders available.

	One example for the CPU clock speed slider to 2.4Ghz			
7	Change the values of the available dropdown menus to filter out some options Note: this test will be done for all the types of parts, so data will change. One example for the RAM type dropdown to "DDR3"	This test is dependent on the range of parts and dropdown menus available	When the search button is clicked only the parts that fit the criteria given or better are displayed.	This to make sure the appropriate parts the user wants are displayed when they input multiple perimeters using the dropdown menus available.
8	Click the Reset Button	Valid	When the reset button is clicked, all the filter values are set to their default values.	This is to ensure that the filters are set to their default values when the user clicks on the reset button. This is so the user does not have to reset each filter manually to display all the parts available.
9	Select another type of part from the parts dropdown menu. (E.g "Storage")	valid	When a new type of part is selected the appropriate dropdown menus and sliders are activated for the user to filters through the new list of parts.	This test is to make sure that the required sliders and dropdown menus are displayed for each PC part type when they have been selected

Post Development (Final) Testing

This is the final test of the solution and should test all the features of the solution to ensure they all work cohesively together. This will mean redoing the tests mentioned in the iteration tests. The final test should not require any additional data but some test may be done prior to the final test for any additional iterations. For my final test I will be using the app as a new user, with no previously saved builds. I will also use all the features in the series of tests listed below.

Test number	Test	What should happen	justification
1	Navigation test: Click on the Build Page, Home Page and Selection Page buttons	The appropriate pages are displayed as I click each button.	This is to ensure that the user can navigate between the various pages of the app seamlessly.
2	Parts selection test: Add a part to the PC build using the filtering various	The filtering system eliminates the parts that do not fit the criteria I give to the filtering system. Then when I select the part	This is to test the parts filtering system to see if it works smoothly and correctly. This also is to ensure that PC parts can be added to the PC build and the information about the PC Build will be updated to an accurate value because of it.

	elements of the filtering system.	I want, it is displayed on the parts selection screen.	
3	Removal part: Remove the part that was added in the previous test	When parts are selected to be removed they no longer appear on the PC build page.	This is to ensure that the user will be able to remove the parts they do not want to be in the PC build and the information about the PC Build will be updated to an accurate value as a result of it.
4	Ease of building test: Create a PC with a multiple types of parts.	Each part I add will be added to the PC build.	This is to make sure that the process of adding and removing parts can be done with multiple parts of different kinds that are involved in the same build.
5	Save PC Build test: Add a name (e.g "My first build") for the build and click the save button.	When I click the save button the parts I have selected are saved under the name I give it and the name will appear on the saved build list.	This is to ensure that the user can save the PC build for later usage.
6	Save multiple PC Builds test: Create another PC and save it under a different name (e.g "Alan's PC build").	When I create more PC builds and save them under different names, their respective names will appear on the saved builds lists.	This is to ensure that the user can save multiple PC builds for later usage. In addition, to have saved PCs for me to compare with each other to test if the load function works properly.
7	Load a PC build test: Load the first build back (the build named "My first build").	When I click a button to load a saved PC build, its parts will be displayed in the saved PC build page.	This is to ensure that the user can load a saved PC build to view or make changes to it.
8	Updating a saved build test: Create a new build and name it after PC that has already been saved (e.g "Alan's PC build") and save it.	When I click the save button the new set of parts from the current PC build will replace the parts in the saved PC build of the same name.	This is to make sure that changes can be made to a saved build will be saved under the same name but duplicates are not made as a result.
9	Updated saved build Check: Update build Load a previously saved build and then load the build that was updated.	When I load the PC build I updated in the previous test, the parts that comprise that PC build is displayed on the PC build page.	This is to make sure when the PC that was updated is loaded the correct PC parts are displayed.
10	Delete a Build test: Click the delete button next to a saved build.	When I click the delete button the selected PCs are removed from the saved builds list.	This is to make sure that the user can delete builds they no longer want to save.

11	Close test: Click the quit button the and reopen the app.	When I close the application using the quit button and then reopen it, all the saved builds are still available to be loaded to the PC build page.	This is to make sure the user can come back to the app after closing it and still be able to interact with their collection of saved PC builds that had been saved before closing the app.
----	--	--	--

Design bibliography

I will now begin the development of my solution using the Unity game engine, scripting in C# and using SQLite for database management. Here are references to the APIs I will use to assist my development.

Codecademy. (2017). LIST OF SQL COMMANDS.

Available: <https://www.codecademy.com/articles/sql-commands?r=master>.

Last accessed 22nd Dec 2017.

Unity Technologies. (2017). SCRIPTING API.

Available: <https://docs.unity3d.com/ScriptReference/index.html>.

Last accessed 22nd Dec 2017.

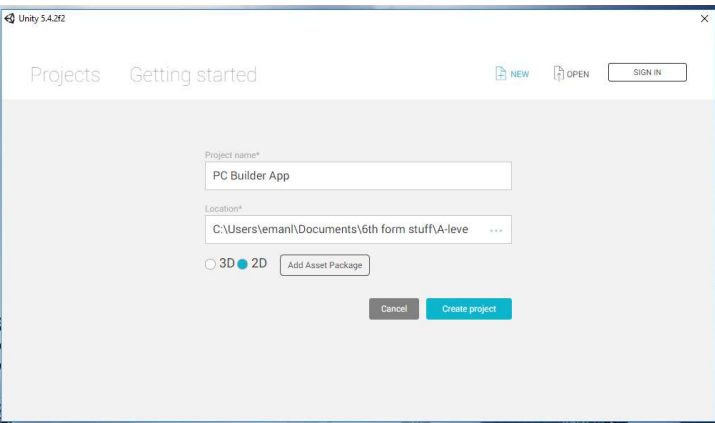
Development

Setting up the development environment

Before I start, implement my design I need to first set up the programing environment and file management structures that I will use.

Unity Engine

To begin development I need to create a new Unity project. This is done by clicking the “NEW” button in the top right of the unity window that first appears when the engine is opened (figure 1).	 <p>fig.1</p>
--	---

Then I assign a project name and a save location for the project. I have chosen a project name of “PC Builder App”. I have set this to a 2D project because it requires less processing power and I do not need any specific features from the 3D engine that I cannot get from the 2D engine. I have not added any assets pages. (figure 2)	 <p>fig.2</p>
--	---