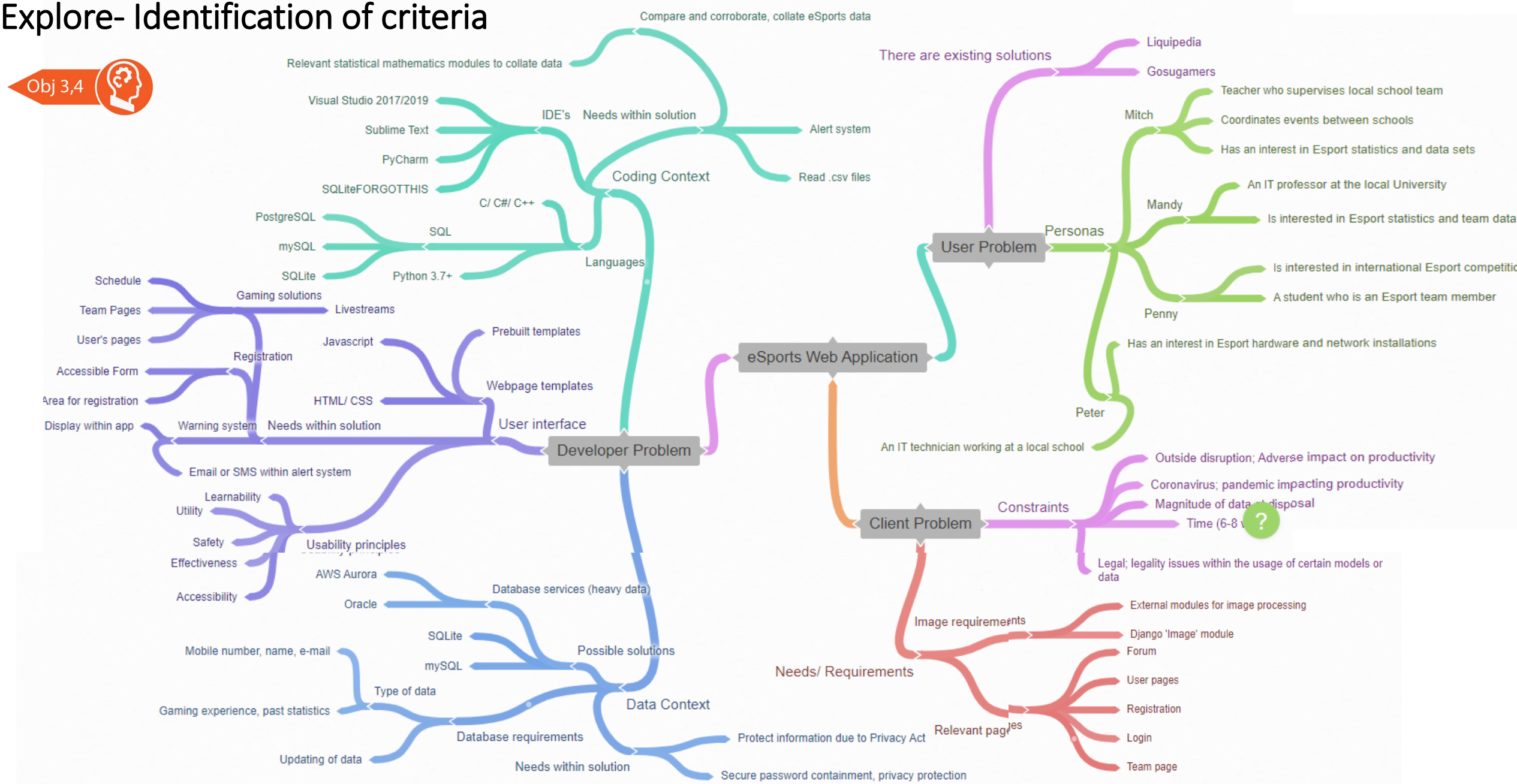
Digital Solutions

Bundaberg eSports FIFA Web App
IA2 Project; Prototype

Paddy Maher

Explore- Identification of criteria

Obj 3,4



Development- Solution overview and Criteria

Obj 1,2 | Obj 3,4 | Obj 8

PC(n) and SC(n) refer to the nth element of either the Prescribed Criteria or Self Prescribed Criteria

INPUT

PROCESS

OUTPUT

Prescribed Criteria;

1. Recognise and describe programmed and UI components with accessibility, useability, effectiveness, safety, utility and learnability
2. Symbolise developer problems using mind maps or ER diagrams, interrelationships between user experiences and data
3. Explain internal and external data components using appropriate symbols, code and data samples. How elements in user-interface connect in annotated diagrams
4. Analyse prototype web application and information to identify data inputs, data and programmed components and their relationships, prototype's personal, social and economic impacts
5. Determine solution requirements that include; essential elements on useability principles, data structures and linkage to interface and code
6. Synthesise ideas and information about solutions for UI, data and programmed components of the prototype, data repositories and programming to generate a prototype web app
7. Generate sample code for 4-6 A4 pages demonstrating selection, iteration, user input and data output- a prototype web application by combining the UI, data and coded components
8. Evaluate against criteria personal, social and economic impacts supported by a collection of data samples or representations. Accuracy and efficiency of coded components in tables, diagrams and paragraphs
9. Make refinements and recommendations for current and future improvements

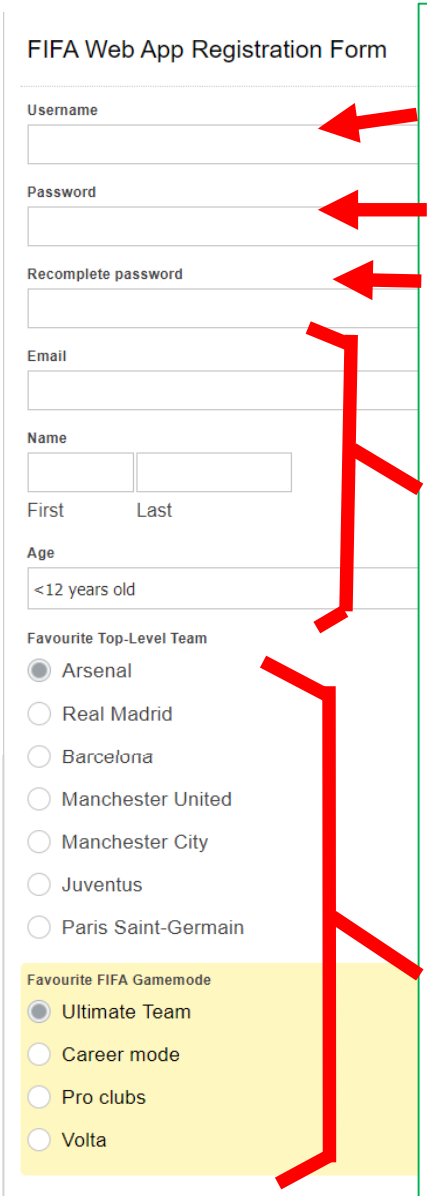
Self-Prescribed Criteria;

1. Compile relevant eSport into data repositories to be accessed through a prototype web application
2. Gain insight on the audience within FIFA eSports and how visual communication elements can be manipulated to optimise traffic for a web app
3. Understand how existing solutions have developed their web apps with relevant code and UI components
4. Generate a web app using relevant front and back-end codes using web frameworks such as Django
5. Adapted for the Bundaberg community
6. Alert system including notifications and emails

Constraints

- Time; 6-8 Weeks
- Legal; components within data usage may have legality issues
- Coronavirus; pandemic impacting productivity
- Outside disruption; adverse impact on productivity

Development- User Interface



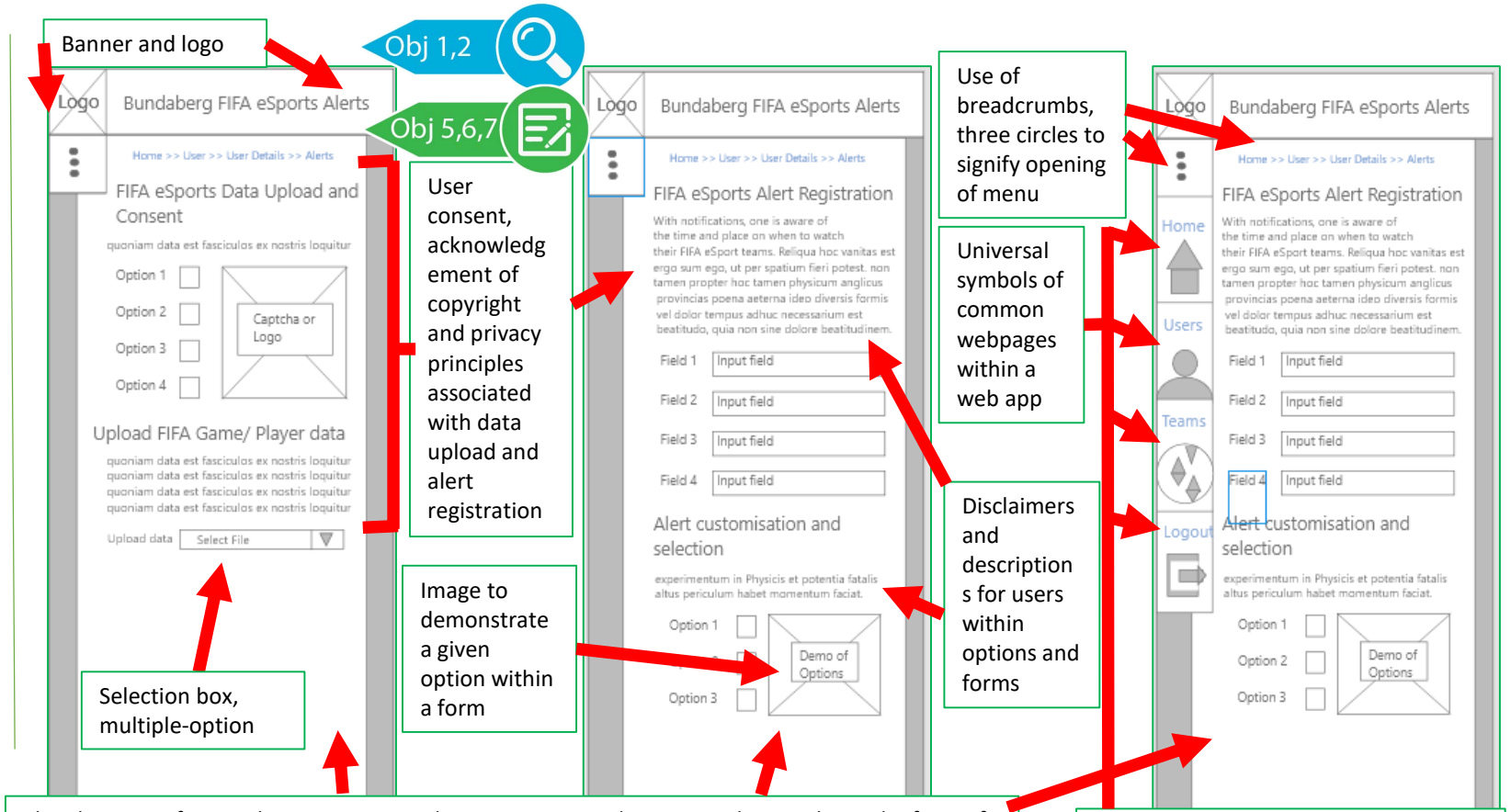
This form was created with wufoo (https://www.wufoo.com/)

Banner and logo with username and password fields. These fields use the visual communication element of alignment with the visual communication principle of proximity.

The retyping of the password is common within registration pages as retyping the password strengthens the memory the user will have with the password.

Email and First name are also common, and can link a person to an accounts. This is particularly useful for eSport professionals, who wish to spread their name and can get contracts through e-mails, thus making the FIFA Web app more suitable for professional FIFA eSports, Age is also important for professional eSports and may be used for safety functions. Age, unlike the previous fields use a scroll=down menu for options.

Multiple-choice fields are used within characterisation for creation of the FIFA follower's account. Options such as 'favourite top team' and 'favourite FIFA gamemode' give the option of using algorithms for users to access an environment more suited to their likes and with people with similar tastes, thus making it more likely for users to stay on the site.

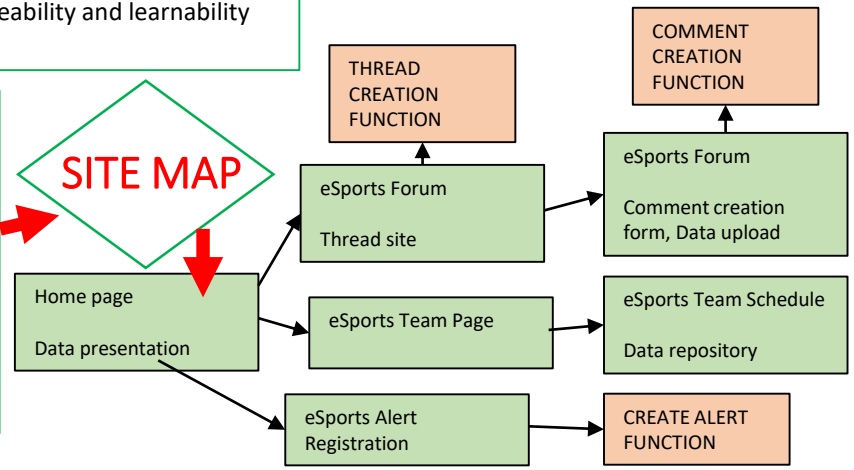


The above wireframes demonstrate visual communication elements and principles in the form of alignment and placement of the entities within the page. Boxes for entering fields and affirming Boolean values are evenly spaced throughout the page, attributing to useability and learnability within the page

Text coloured blue indicates hyperlinks

Through creating templates with elements of breadcrumbs, site maps are integral in indicating the paradigm of the web app. Whilst there are more to added to this site map seen to the right, this site map indicates all front-end elements that hold functions. Understanding the relationships between entities is important within the developer view, and helps to understand how a user interacts within the app.

Outlining the site map also indicates the significance of navigation throughout a given web app, as the greater the complexity of the site map, the more useability and learnability techniques should be used



Development- User Interface and Data

```
DELETE FROM User
WHERE 'id' = 1
```

```
INSERT INTO User VALUES(
25,'Frederich', 'FIFA Team 1','13',
'Beyond_Good_And_Evil@outlook.com'
);
```

The 'CREATE' SQL statement is used for creating a model within an SQL server. Within an SQL statement, fields are initialised first through their names, and then characterised through various keywords that describe their data type. For example, the 'username' field is written with the name of the field 'username', followed by it's datatype 'VARCHAR', with the amount of text within the field (20), followed by the requirement for the data not to be null.
https://www.w3schools.com/sql/sql_create_table.asp
Other statements can be used to manually insert and remove entries and fields within a table such as 'DELETE' and 'INSERT INTO'.
(INSERT INTO)
https://www.w3schools.com/sql/sql_create_table.asp,
(DELETE)
https://www.w3schools.com/sql/sql_delete.asp


```
CREATE TABLE User {
id integer NOT NULL UNIQUE PRIMARY
AUTOINCREMENT,
username varchar(20) NOT NULL,
team_following varchar(30) DEFAULT
NULL,
wins integer DEFAULT NULL,
email varchar(30) NOT NULL,
};
```

This data has been checked for data integrity in order to prevent data redundancy.
Redundant data can come in the form of;
- Two fields being the same
- Data being unreadable or not within format of the SQL table (input constraints)
- Null values in fields that are not meant to be null
- Data types different to that specified within the field

For a developer working through SQL within web apps, SQLite is a light-weight iteration of SQL similar to MySQL and PostgreSQL thus making it appealing particularly within prototyping databases. Within development of SQLite servers, SQLite Database Browser provides a way to access these databases with easy implementation of SQL commands.
https://sqlitebrowser.org/

The MySQL and SQLite use AUTOINCREMENT. This is used as it indicates the database automatically creates these fields within a given model as default. Usually this refers to the 'id' of a given object.

https://www.w3schools.com/sql/sql_autoincrement.asp

The various data types within this table are;

```
'ID':INTEGER
'THREAD_TITLE':VARCHAR
'THREAD_TEXT':VARCHAR
'PUBDATE':DATETIME
'UPVOTES':INTEGER
'USER_ID':INTEGER
```

The table as seen on the bottom right is the '**main/thread**' (<< NOT THE NAME CHANGE) table and is used for displaying the threads within the FIFA eSports Forum. Users register data in this table through forms and fields are inputted by users such as 'thread_title' and 'thread_text'. However fields such as 'id','user_id' and 'pub_date' are completed automatically on creation of the thread object.

The threads for displaying are called using a Foreign Key to the Thread model.

Foreign Keys will be used also for other functional components necessary to an eSports app such as an alert system. The implementation of these Foreign Keys and not keeping all models and information confined to one table retains data integrity.

Data within this table will be deleted in seven days. There are various reasons;
- Seven days is enough to collect a week of FIFA eSports discussion and is useful for information to optimise the Web app's function
- The data cannot stay indefinitely to retain the privacy of the users
- The website and development could be updated and there is a risk the data will become redundant
- Most recent data is typically most relevant as it describes the current interests of the FIFA Web app user's interests

	id	thread_title	thread_text	pub_date	upvotes	user_id
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Best player in FIFA 20?	Thinking alon...	2019-07-29 0...	0	1
2	2	premier league thread	all things prem	2019-08-04 0...	0	5
3	3	TOTS Lacazette OP?	Been facing hi...	2019-08-05 0...	0	4
4	4	Ligue 1 or Serie A?	??	2019-08-05 0...	0	6
5	5	Best Starting 11 Thread	Looking to pro...	2019-08-11 0...	0	6
6	6	Ultimate Team Tweaks	Discuss	2019-08-11 0...	0	4
7	7	FIFA Update Discussion...		2019-08-11 0...	0	2
8	8	eSport FIFA League dis...		2019-08-17 0...	0	6

Development- Data 1

Obj 3,4

Obj 5,6,7

	A	B	C	D	E	F	G	H	I	J
1	sofifa_id	player_url	short_nan	long_nam	age	dob	height_cm	weight_kg	nationalit	club
2	158023	https://so	L. Messi	Lionel An	32	########	170	72	Argentina	FC Barc
3	20801	https://so	Cristiano	Cristiano	34	########	187	83	Portugal	Juvent
4	190871	https://so	Neymar Jr	Neymar d	27	########	175	68	Brazil	Paris Sa
5	200389	https://so	J. Oblak	Jan Oblak	26	########	188	87	Slovenia	AtlÃ©t
6	183277	https://so	E. Hazard	Eden Haza	28	########	175	74	Belgium	Real M
7	192985	https://so	K. De Bruy	Kevin De	28	########	181	70	Belgium	Manch
8	192448	https://so	M. ter Ste	Marc-And	27	########	187	85	Germany	FC Barc
9	203376	https://so	V. van Dij	Virgil van	27	########	193	92	Netherlar	Liverpo
10	177003	https://so	L. ModriÄ	Luka Mod	33	########	172	66	Croatia	Real M
11	209331	https://so	M. Salah	Mohamed	27	########	175	71	Egypt	Liverpo
12	231747	https://so	K. Mbapp	Kylian Mb	20	########	178	73	France	Paris Sa
13	201024	https://so	K. Koulib	Kalidou K	28	########	187	89	Senegal	Napoli
14	202126	https://so	H. Kane	Harry Kan	25	########	188	89	England	Totten
15	212831	https://so	Alisson	Alisson Ra	26	########	191	91	Brazil	Liverpo

CSV

There are methods of exporting .csv to SQL or SQLite such as with USBWebserver and phpMyAdmin

https://www.sqlshack.com/importing-and-working-with-csv-files-in-sql-server/

This diagram describes how the models relate to each other with Foreign Keys and particularly one-to-many relationships.

THREAD

PK; **THREAD_ID**; INTEGER,
THREAD_TITLE; VARCHAR(30),
THREAD_TEXT; VARCHAR(50),
PUBDATE; DATETIME,
UPVOTES, INTEGER,
USER_ID, INTEGER

COMMENT

PK; COMMENT_ID,; INTEGER,
COMMENT_TEXT; VARCHAR(50),
PUB_DATE; DATETIME,
UPVOTES; INTEGER,
THREAD_ID; INTEGER,
USER_ID; INTEGER

USER

PK; **USER_ID**; INTEGER,
PASSWORD; VARCHAR(50),
USERNAME; VARCHAR(30),
EMAILL VARCHAR(40),
IS_STAFF; BOOLEAN,
IS_ACTIVE; BOOLEAN,
DATE_JOINED; DATETIME,
FIRST_NAME; VARCHAR(20),
LAST_NAME; VARCHAR(20),
IS_SUPERUSER; BOOLEAN

ALERT

PK; ALERT_ID,; INTEGER,
ALERT_TEXT; VARCHAR(50),
PUB_DATE; DATETIME,
USER_ID; INTEGER

And if data is viewed to be redundant, it can be deleted through SQL;
DELETE FROM TABLE Users
FIELD 'is_Staff';
(https://www.w3schools.com/sql/sql_delete.asp)

This table outlines the 'User' model. Within this model, various datatypes are inputted ranging from Booleans (for 'superuser', 'is_staff', etc), varchar (for various fields such as username and email), integer ('id')m and datetime ('date_joined').

Data can be stored within Excel .csv files and through data upload methods, large sums of data can be transferred. This also accounts for downloading open-source csv files for easy data that can help optimise the FIFA Web App.

```
ALTER TABLE Users
ADD
    FIFAWins integer,
    FIFALosses integer;
```

Related eSport components can always be added through the SQL 'ALTER' statement.
http://w3schools.com/sql/sql_alter.asp

Data can be inputted within this SQL server through insert commands (https://www.w3schools.com/sql/sql_insert.asp#:~:text=INSERT%20INTO%20Syntax,%2C%20column3%2C%20...))

	id	password	last_login	is_superuser	username	first_name	email	is_staff	is_active	date_joined	last_name
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	pbkdf2_sha25...	2019-08-06 0...	1	paddy	Patrick	maherp1@sh...	1	1	2019-07-29 0...	Maher
2	2	pbkdf2_sha25...	2019-08-25 0...	0	FIFAFan2020			0	1	2019-08-04 0...	
3	3	pbkdf2_sha25...	2019-08-04 1...	0	Barcelona7nil			0	1	2019-08-04 1...	
4	4	pbkdf2_sha25...	2019-08-24 0...	0	FIFA15Best			0	1	2019-08-05 1...	
5	5	pbkdf2_sha25...	2019-08-29 1...	0	Lacazette20	test	testaccount@...	0	1	2019-08-06 0...	account
6	6	pbkdf2_sha25...	2019-08-31 0...	1	admin		maherp1@sh...	1	1	2019-08-11 1...	
7	7	pbkdf2_sha25...	2019-08-18 0...	0	TheFIFAPro	test2	test2@outlook...	0	1	2019-08-18 0...	test2
8	8	pbkdf2_sha25...	2019-08-29 1...	0	SweatyGoals50	third	test3@outlook...	0	1	2019-08-18 0...	test
9	9	pbkdf2_sha25...	2020-05-28 0...	0	beyondgooda...	Fredriche	beyondgooda...	0	1	2020-05-28 0...	Nietzche

```
INSERT INTO
User VALUES(
'password',
'0',
'beyondgoodandevil',
'Fredriche',
'beyondgoodandevil@outlook.com',
'0',
'Nietzche');
```

Development- Data 2

A pivotal SQL command is the creation of the app through the 'Create database' command

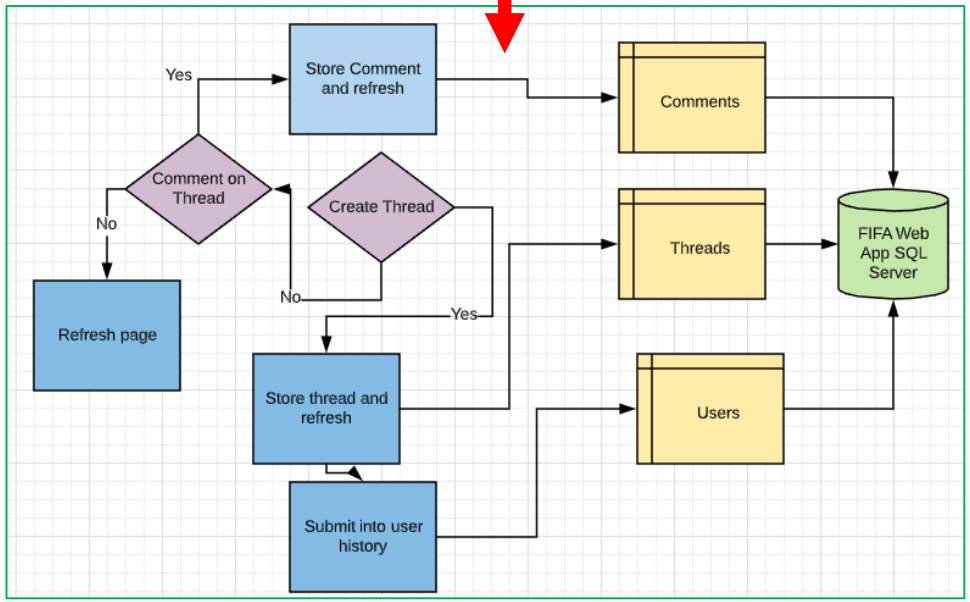
```
Create database
"BundabergFIFAeSports"
```

This code describes how FIFA components can be inputted to the table with values following this. 'ALTER' is often using more than 'CREATE' due to constant changing in development and generation of a given digital solution as described by the EDGE process

```
ALTER TABLE
main_additionaluseri
nfo {
"FifaGoals" integer
DEFAULT = "0",
"FifaYellowCards"
integer,
"FifaRedCards"
integer,
"FifaLeaguesWon"
integer};

INSERT INTO TABLE
main_additionaluseri
nfo
VALUES {'5',
'UltimateTeam',
'2',
'[Image]',
'23',
'50',
'2',
'6'};
```

The below ER Diagram describes the logical flow of various systems within the Bundaberg FIFA eSports Web app in particular the Forum system. Through decisions on events such as creation of threads and comments can different events be fired and tracked to internal storages and eventually the SQLite server




```
CREATE TABLE main_additionaluserinfo {
"id" integer NOT NULL PRIMARY KEY
AUTOINCREMENT,
"pc" integer NOT NULL,
"user_id" NOT NULL UNIQUE REFERENCES,
"image" varchar(100) NOT NULL};

CREATE TABLE  main_thread { "id"
integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "thread_title"
varchar(50) NOT NULL,
"thread_text" varchar(200) NOT NULL,
"pub_date" datetime NOT NULL,
"upvotes" integer NOT NULL, "user_id"
integer NULL REFERENCES
"auth_user"("id") DEFERRABLE INITALLY
DEFERRED};

CREATE TABLE main_comment { "id"
integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "comment_text"
varchar(20) NOT NULL,
"pub_date" datetime NOT NULL,
"upvotes" integer NOT NULL,
"thread_id" integer NULL REFERENCE
"main_thread"("id") DEFERRABLE
INITIALLY DEFERRED,
"user_id" integer NULL REFERENCES
"auth_user"("id") DEFERRABLE INITALLY
DEFERRED};

CREATE TABLE auth_user {"id" integer
NOT NULL PRIMARY KEY AUTOINCREMENT,
"password" varchar(128) NOT NULL,
"last_login" datetime,
"is_superuser" bool NOT NULL,
"username" varchar(150) NOT NULL,
"first_name" varchar(30) NOT NULL,
"email" varchar(254) NOT NULL,
"is_staff" bool NOT NULL,
"is_active" bool NOT NULL,
"date_joined" datetime NOT NULL,
"last_name" varchar(150) NOT NULL};
```

The following 'CREATE' SQL tables command contain four of the prominent models in the Bundaberg FIFA Web App. The first of these demonstrated is "main_additionaluserinfo" table. A custom-made table for adding to the 'User' modelIt's first field is autogenerated (as indicated by the 'AUTOINCREMENT' statement) by Django and SQLite as the 'id' field. This is found within each of the tables as the primary key. The "user_id" in addition to being 'NOT NULL' is registered as 'UNIQUE REFERENCES' indicated this field is referenced within other models. This referencing is integral to formation of foreign keys and is unique in Django as the convenience for creation of foreign keys is notable.

Following the "main_additionaluserinfo" table is the basis of the Bundaberg FIFA Web App's Forum, "main_thread". Similar to the "main_additionaluserinfo" table, the "main_thread" is custom-generated. It contains two text options in the title and text allowing all characters via the 'varchar' datatype. 'datetime' is used for the publication date field 'pub_date'. The thread is linked back to the user using the 'REFERENCES' to the table "auth_user", requesting the field "id" for the "user_id". This is initially deferred via the SQL statement "DEFERABBLE INITIALLY DEFERRED"

In serving a similar purpose to the 'thread' model in the paradigm of the web app, the 'comment' model subsequently has a similar data store. It retains most of the same fields as 'main_thread' despite not having a title, however 'main_comment' links to the thread itself as a foreign key. Similar to how 'main_thread' links to the 'User' model via the 'user_id' and it's table 'auth_user', 'main_comment' links to the 'Thread' model via the 'thread_id' and it's table 'main_thread'. It also links back to the user, allowing users to upvote and create threads and comments alike.

The likely most prominent model initialised is the 'User' model. Unlike the previous three tables ("main_additionaluserinfo", "main_thread", "main_comment") which were custom-made, the 'User' model is a Django generated table. This difference in development is reflected through the SQL table. Using a different datatype as seen from before, "bool" can be seen to be used for three fields ("is_superuser", "is_staff", "is_active"). Bools are binary functions in that they can only be true or false. Although "is_superuser" and "is_staff" can seem to be similar and do serve similar purposes within Django admin, Django provides more choices to developers on which field to use. Many other standard fields for User creation can be seen in this table such as "password", "username", "email", "first_name" and "last_name". However it must be noted that (as in the previously data slide's table representing this SQL table) the password data is encrypted.

Development- Algorithms

Algorithm for processing Threads

Obj 1,2

Obj 8

```
SELECT FROM TABLE 'main.threads'
FOR EACH thread;
 IF 'thread.user' banned THEN
   NOT display thread
 ELSE IF 'thread.title' NOT appropriate THEN
          NOT display thread
          ELSE IF 'thread.text' NOT appropriate THEN
                    NOT display thread
                     ELSE  display.thread
```

Algorithm for triggering Alerts

```
Create list of events within FIFA Web app that has cause to
trigger a user;
LIST : [New_FIFA_Update, FIFA_Web_App_Update,
Force_Update, User_Banned_or_Suspended]

Check if a given event is within the list
IF event IN LIST THEN
    CREATE NewAlert
    IF event = {New_FIFA_Update} SET warning LOW
    ELISEIF event = {FIFA_Web_App_Update} SET warning
MEDIUM
    ELSEIF event = {Force_Update,
User_Banned_or_Suspended} SET warning HIGH
```

The registration algorithm begins by gathering the data from the UserForm. The algorithm goes on to check whether this data is double data (already within database) and inputs to the database if it tests negative to already being in the database.

This algorithm processes the creation and the validation of the 'Thread' model, that is the key model for the discussion forums Within this pseudocode algorithm, conditional statements 'if' and 'then' check if the user is banned or if the content they have provided is appropriate

The FIFA Web app comment algorithm is similar to the thread algorithm, checking if the user is banned and if the comment's data (text and image) is appropriate. In the case the data is appropriate and the user is not banned, the comment is outputted.

The FIFA Web app employs an alert system. Using a list of possible events that warrant an alert, the algorithm deciphers whether the event permit an alert using a conditional statement. An alert is created with varying levels of significance, represented by warning level, depending on which event is inputted to the algorithm.

Algorithm for registration (Add Users)

```
GET * FROM UserForm
IF username IN TABLE Users
          THEN Redirect
ELSE
          INPUT INTO TABLE Users { username, password,
          email, first_name, last_name }
          LOGIN USER
          Redirect
```

Algorithm for processing Comments

```
 SELECT FROM TABLE 'main.comments'
FOR

FOR EACH thread;
 IF 'comment.user' banned THEN
   NOT display thread
 ELSE IF 'comment.image' NOT appropriate THEN
          NOT display thread
          ELSE IF 'comment.text' NOT appropriate THEN
                    NOT display thread
                     ELSE  display.thread
```

Algorithm for FIFA ranking

```
IF User.Goals <10 THEN Rank = 'Bronze'
ELSEIF User.Goals <50 THEN Rank = 'Silver'
ELSEIF User.Goals <200 THEN Rank = 'Gold'
ELSEIF User.Goals <500 THEN Rank = 'Platnium'
ELSEIF User.Goals >500 THEN Rank = 'Masters'
```

Through the submission of information by the user such as their favourite Football/ Soccer Team, algorithms such as this match these users to other users of the same favourite team, thus they will find the site more user-friendly to their interests. This algorithm iterates through users and inputs them into tables characterized by their favourite team. More algorithms could be used to use this data to increase the probability of users in each table to interact with eachother.

Algorithm for matching interests

```
GET * FROM User TABLE
          FOR user in User TABLE:
          GET 'favourite_team' from user
                    INPUT user INTO
          {'favourite_team'} TABLE
```

Algorithm for validating email

```
IF User.Registered = True
GET FROM User TABLE User.Email
                    SEND Verification_Email
                    to User .Email
```

Within the FIFA eSports Web app, emails will be sent in order to test the validity of the email inputted by a user registering in the UserForm.

Similar to matching interests, users can also be categorized by their skill in FIFA, in this case being determined by the amount of career goals scored. This algorithm retrieves the user's amount of goals and sets a list of goal requirement for different ranks varying from 'Bronze' to 'Masters'.

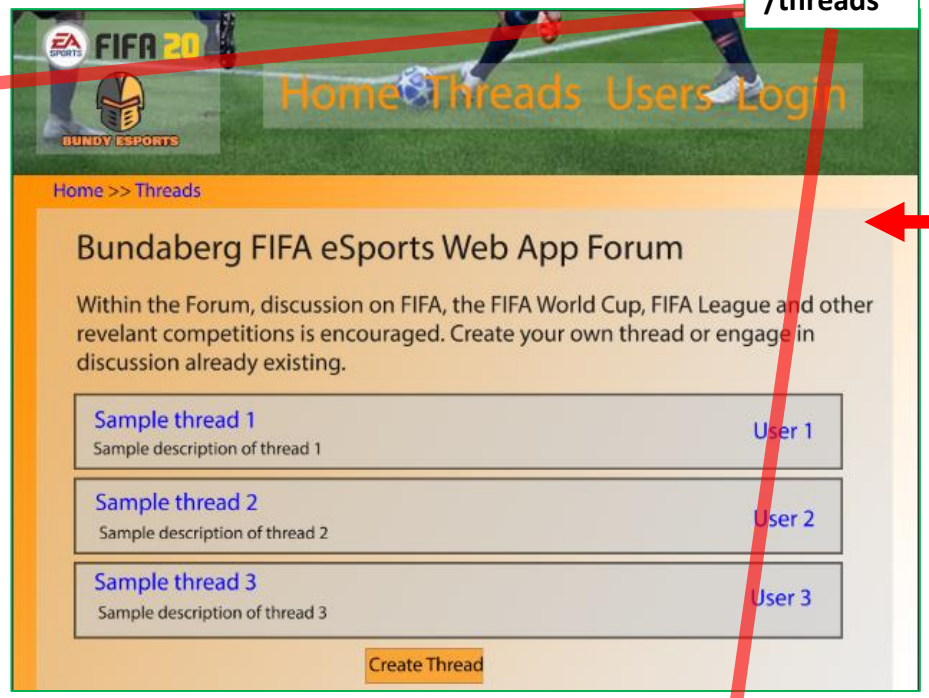
Generate- User interface (HTML code)

/threads/Sample_thread_1.html

This HTML page demonstrates the view inside one of the threads. As seen here the 'thread_title' and 'thread_text' seen in the URLS '/threads' and '/threads/CreateThread.html' are at the top of the page. Below this are comments, with their respective comment texts and the user linked. A reply to a comment can also be seen within the first sample comment.

The navigation bar is integral for the useability and learnability of the web app. The leftmost of the navigation shows the FIFA 20 Logo and the Bundaberg eSports logo. To the right of this are the relevant main pages with a backdrop of a FIFA image. The translucent backdrops of the texts and logos make the text readable without obstructing on the image.

/threads

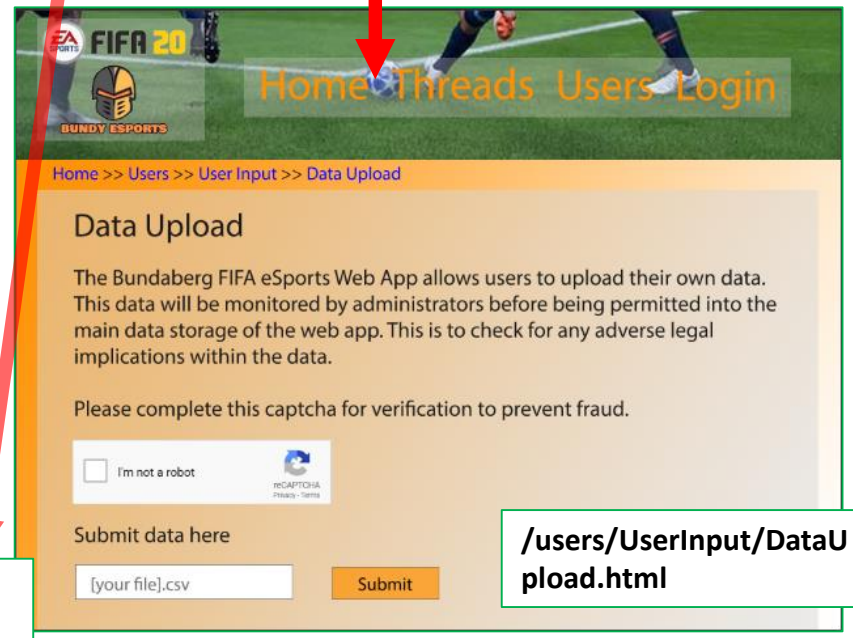


The '/threads' page is the basis of the Bundaberg FIFA eSports web app forum which consequently has mass social, economic and personal impact on the Bundaberg FIFA community. A title followed by a description is listed at the top in black to contrast the orange and white gradient background. Following this, thread models are displayed via boxes with the blue thread titles indicating links. Alongside this is the thread description and the user whom created the thread, which also has a link. The thread page also has an option to create a thread which will link to the ThreadForm through the URL '/threads/CreateThread.html'

The thread creation form uses alignment for fields combined with a captcha for data security.

/threads/CreateThread.html

The data upload is important for collecting FIFA data and thus making the site better for this. A captcha is used to prevent mass botting of data upload. The data is accepted at the bottom through .csv files. A disclaimer is also given stating moderators will check data before officially uploading it to the main database.



/users/UserInput/DataUpload.html

Generate- Code

Obj 1,2 | Obj 3,4 | Obj 5,6,7

Django is a Python Web framework for quick development and automates much of the process for web development whilst still giving access to many high-level python functions. Considering the personas listed within the 'Technical Proposal', it can be seen Django is adequate for all needs of the personas. Mitch, being a teacher supervising a team and interested in Esport statistics can find interest in the manipulation and presentation of the statistics for Esports through Django forms and views. Penny, an Esport team member, can find aspects of the Web app developed within Django such as the Forum and the User Profile listing statistics useful. Mandy should find interest in the data collected throughout discussion forums and Esport User Profiles, where FIFA games are outlaid. Peter will find interest in the code and what hardware can run Django software, the modules and external models being inputted within this model and what networks are being installed throughout this system.

Modules are initialised within the top of the python file for the necessary files and external Esport modules to satisfy the needs of the audience and provide enough functional aspects for an Esport web app

Views.py

```python
from django.shortcuts import get_object_or_404, render, redirect
from django.views import generic
from django.template import loader
from django.http import HttpResponse, HttpResponseRedirect
from django.views.generic import DetailView, ListView
from .models import *
from django.utils import timezone
from django.urls import reverse
from .forms import (CommentForm, ThreadForm,
RegistrationForm, EditProfileForm, AdditionalUserInfoForm)
from django.views.decorators.http import require_POST
from django.contrib.auth.forms import PasswordChangeForm
from django.contrib.auth.models import User
from django.contrib.auth import login, logout, authenticate, update_session_auth_hash
from django.contrib import messages
```

A template and parameter is initialised via the URL and the 'threadset' for an Esports forum. A function returns all Esports Thread objects for HTML code to output and context is gathered for the Esports Thread.

Python classes ('w3schools.com/python/python_classes.asp') are used within Django for 'Models', as they are used traditionally for objects. The HomeView is listed as a class and its parameter links the type of view via the 'generic' format made by Django which are essentially class-based views ('docs.djangoproject.com/en/3.0/topics/class-based-views/generic-editing/')

```python
class HomeView(generic.ListView):
    template_name = 'main/home.html'
    context_object_name = 'threadset'

    def get_queryset(self):
        return Thread.objects.all()

    def get_context_data(self, **kwargs):
        # Call the base implementation first to get a context
        context = super().get_context_data(**kwargs)
        context['form'] = ThreadForm()
        return context
```

The view following the 'HomeView' is constructed differently. Appeasing to personas as to the complexity of the solution, using separate functionalities. As opposed to a class-based view of HomeView, the 'detail' view is a function-based view ('docs.djangoproject.com/en/3.0/topics/http/views/'). The parameters are the HTTP request, and the 'id' for any given thread, as the detial view views a specific thread. The object is retrieved using 'get_object_or_404' with parameters of the Thread model and specifically selecting one with the thread_id as given by the function parameters. A comment form is initialised, context is given for HTML code and the page is outputted using Django's render function ('docs.djangoproject.com/en/3.0/topics/http/shortcuts/')

```python
def detail(request, thread_id):
    thread = get_object_or_404(Thread, pk=thread_id)

    form = CommentForm()

    context = {
        'thread': thread,
        'form': form
    }
    return render(request,'main/detail.html',context)
```

The ResultsView is a second class-based view and outputs pure Esports data and results, appeasing personas such as Mandy and Mitch who are interested in Esport statistics and data sets. Its model is initialised and the template is given as a HTML file using a 'DetailView'

```python
# 'ResultsView' irrelevant currently
class ResultsView(generic.DetailView):
    model = Thread
    template_name = 'main/results.html'

@ require_POST
def addComment(request, thread_id):
    form = CommentForm(request.POST)

    if form.is_valid():
        new_comment = Comment(comment_text=request.POST['text'],
            pub_date=timezone.now(),
            thread=get_object_or_404(Thread, pk=thread_id), user=request.user)
```

The 'addComment' function is less of a function-based view as it is a function for formatting comments for other views such as the 'detail' view. The '@ require_POST' highlights the purpose for this function, as it is ran only when a user posts data into the system via the HTTP request 'POST'. The form is initialised as the CommentForm with the parameter as the POST request, where users can create Comments on a given Esports Thread. The form is checked for validity via the 'form.is_valid()' Django function (docs.djangoproject.com/en/3.0/ref/forms/validation/'). 'new_comment' is initialised as the comment posted with the pub_date, thread, primary key and user initialised for the entity.

The comment is saved for the Esport thread and the user is redirected to the thread the comment was created on with the page refreshed with new comments.

Obj 8

```python
        new_comment.save()

    return redirect('main:detail', thread_id)

@ require_POST
def addThread(request):
    form = ThreadForm(request.POST)

    if form.is_valid():
        new_thread = Thread(thread_title=request.POST['title'],
            pub_date=timezone.now(),
            thread_text=request.POST['text'],
            user=request.user)
        new_thread.save()

    return redirect('/threads')
```

Following the format of the 'addComment' function, 'addThread' works in a similar fashion, initially requiring a HTTP Post request for the function for creation of an Esports discussion thread. The form for thread creation is initialised with the form.is_valid() statement as in the addComment function. Initalising the fields of the Esports Thread model, the object is then saved. The user is then redirected to the threads page.

The register function-based view is integral within any Esports web app for the creation of profiles. Two Esport forms are initialised within the function being the Django provided 'RegistrationForm' ('django-registration.readthedocs.io/en/2.0.4/forms.html'), alongside the 'pc_form' a custom made 'Preferred Category' form both with the POST request as their parameter. Both forms are checked for validity using an 'if' statement with various fields being saved using such functions as 'cleaned_data' to retrieve data. The user is then logged in using Django's login function and redirected to the home page.

In the case the data is not valid, the 'else' statement is used. Initialising the two forms and re-rendering the Esports registration form with context initialised.

```python
def register(request):
    form = RegistrationForm(request.POST)
    pc_form = AdditionalUserInfoForm(request.POST)
    if form.is_valid() and pc_form.is_valid():
        user = form.save()

        pc = pc_form.save(commit=False)
        pc.user = user
        pc.save()

        username = form.cleaned_data.get('username')
        messages.success(request, f"New account created: {username}")
        login(request, user)
        return redirect('/')
    else:
        form = RegistrationForm()
        pc_form = AdditionalUserInfoForm()
    context = {
    'form':form,
    'pc_form': pc_form,
    }
    return render(request, "main/register.html", context)
```

The 'view_profile' view synthesises Esport data for a player's profile. The 'Preferred Categories' form is initialised and checked for validity, saved and then linked to the user. The context is formatted for variables include the 'user' (linked to the HTTP request) and the 'pc_form'. The render function is used linking the request, the destination URL and the context.

```python
def view_profile(request):
    pc_form = AdditionalUserInfoForm(request.POST)
    if pc_form.is_valid():

        pc = pc_form.save(commit=False)
        pc.user = request.user
        pc.save()
        return redirect('main:view_profile')
    else:
        pc_form = AdditionalUserInfoForm()

    context = {'user': request.user,
    'pc_form': pc_form}
    return render(request, 'main/localprofile.html', context)
```

The 'vote' function, listed the newest user with the amount of users that had registered currently. Using the 'User' model, the usermodel was initialised. The 'userlength' was found by using the len() python function (https://www.w3schools.com/python/ref_func_len.asp) on the array of objects 'users' found within the 'User' model. The 'newestuser' was found by determining the user which had the most recent 'id' which is equivalent to the 'userlength'. This is initialised via the context to be used within render along other parameters 'request' and the destination URL 'main/homepage.html'

```python
# work on upvotes, migrate and change model
def vote(request):
    user = request.user
    usermodel = User
    userlength = len(User.objects.all())
    newestuser = User.objects.get(id=userlength)
    context = {'user': user,
    'usermodel' : usermodel,
    'newestuser' : newestuser,
    'userlength': userlength,}
    return render(request, 'main/homepage.html', context)
```

Generate- Code

Models.py is an integral python file within construction of the FIFA eSports web app using Django. They distinguish the prime function of python in being an object-orientated language, adequate for distributing entities within an eSports App. Similar to the 'views.py' file, the 'models.py' begins by importing the necessary functions and modules for creating eSports objects within the FIFA app. 'Datetime' is imported first, and is a common python module for initialising date fields (https://www.w3schools.com/python/python_datetime.asp). Various Django modules are imported following the first including models, timezone, NON_FIELD_ERRORS, ModelForm (an eSports form constructed from another python file within this app) and the 'User' Django-created model.

Models.py

The 'Thread' model is the first model created and is integral within the creation of the FIFA eSports Forum. The 'Thread' model is a class with the parameters being 'models.Model'. The fields are created for the thread_title, thread_text, pub_date, upvotes and the user. For the thread_title, it is initialised using Django's model object with 'CharField' for use of characters with a maximum length of the field being 50 characters

```python
import datetime
from django.db import models
from django.utils import timezone
from django.core.exceptions import NON_FIELD_ERRORS
from django.forms import ModelForm
from django.contrib.auth.models import User

class Thread(models.Model):
    thread_title = models.CharField(max_length=50)
    thread_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    upvotes = models.IntegerField(default=0)
    user = models.ForeignKey(User, null=True, on_delete = models.CASCADE, unique = False)

    def __str__(self):
        return self.thread_title
    def published_past(self):
        return self.pub_date <= timezone.now()

class Comment(models.Model):
    comment_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published', default=timezone.now)
    thread = models.ForeignKey(Thread, on_delete=models.CASCADE)
    upvotes = models.IntegerField(default=0)
    user = models.ForeignKey(User, null=True, on_delete = models.CASCADE, unique = False)

    def __str__(self):
        return self.comment_text[:20]
```

The thread function uses 'DateTimeField' within the pub_date field, along with a vote field and using Django's 'ForeignKey' to link to the User model (https://docs.djangoproject.com/en/3.0/ref/models/fields/)

The comment model is constructed similarly, with all fields excluding the 'thread' Foreign Key being identical to the 'Thread' model. The '__str__' function returns the comment_text when the Comment model is being called within string.

Forms.py

Forms are used commonly throughout all eSports web apps, for creating accounts with usernames and passwords from picking eSport teams and categories. Common to all python files within this FIFA web app, modules are called. The Django-generated 'forms' are imported and prove to be integral within the 'forms.py' file. All models are imported with 'from .models import *' indicating that '*' includes all models within the 'models.py'. Django models and specificially User are imported with Django generated forms UserCreationForm and UserChangeForm

```python
from django import forms
from .models import *
from django.db import models
from django.contrib.auth.models import User
from django.contrib.auth.forms import UserCreationForm, UserChangeForm

class ThreadForm(forms.Form):
    title = forms.CharField(max_length=50)
    text = forms.CharField(max_length=200)

class CommentForm(forms.Form):
    text = forms.CharField(max_length=200)

class RegistrationForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = (
            'username',
            'first_name',
            'last_name',
            'email',
            'password1',
            'password2',
            )

    def save(self, commit=True):
        user = super(RegistrationForm, self).save(commit=False)
        user.first_name = self.cleaned_data.get('first_name')
        user.last_name = self.cleaned_data.get('last_name')
        user.email = self.cleaned_data.get('email')

        if commit:
            user.save()
```

The ThreadForm is the form for FIFA eSport followers to create threads for eSport discussion. The parameter is Django's 'forms.Form' (https://docs.djangoproject.com/en/3.0/topics/forms/), with the title and text fields created both having CharField, involving all characters within the field

The CommentForm performs a similar function the ThreadForm and only involves text

The RegistrationForm is important for any eSports follower using this app as it being used for user creation. The parameter is the Django's 'UserCreationForm' with 'email' firstly being initialised as an 'EmailField'. The 'Meta' Class links the model as the User and the necessary fields for the registration of the user.

The 'save' function is used in order to commit to the user's data for the registration of an eSports follower within the FIFA eSports web app. User is initialised using the 'super' python function (https://www.w3schools.com/python/ref_func_super.asp) with the user's first and last name along with their email using self.'cleaned_data'. 'self' being a common python system to refer to one's own model and 'cleaned_data' a Django method of making data more accessible and safe. If the user commits, the user is saved through Django's own save function with the user being returned.

```python
        return user

class EditProfileForm(UserChangeForm):
    class Meta:
        model = User
        fields = (
            'email',
            'first_name',
            'last_name',
            'password',
            )

class AdditionalUserInfoForm(forms.ModelForm):
    class Meta:
        model = AdditionalUserInfo
        fields = ('pc',)
```

EditProfileForm uses Django's UserChangeForm. Within the EditProfileForm class, a 'Meta' class is used to define the model and the various fields for the editing of the model including such eSports necessary fields as 'email', 'first_name', 'last_name' and 'password'. AdditionalUserInfoForm acts similarly to EditProfileForm however it uses non-Django constructed fields as 'Preferred Category' was specifically made for the FIFA eSports Web app as opposed to being automatically generated.

Hypertext Mark-up Language (HTML) is the primary mark-up language for front-end programming for web apps. User interface is crucial within an eSports app for useability and learnability of the site, and most of user interface and the visual communication elements and principles are constructed through HTML and it's variants XML, Json, etc. This HTML file begins by extending from a template html file, a Django implemented function, "https://docs.djangoproject.com/en/3.0/ref/templates/language/"

Within Django HTML, logic statements can be creation through braces and percentage symbols, such as {% if [condition] %}. This can be seen through {% if thread %}, asking if the model rendered to the HTML file is a 'thread' model (as initialised in models.py). The thread_title is header using <h1> tags (https://www.w3schools.com/tags/tag_hn.asp)

```
{% extends "main/header.html" %}

{% block content %}
{% if thread %}
    <h1>{{thread.thread_title}} || <a href='/users/{{thread.user.id}}'>{{thread.user.username}}</a></h1>
    <p>{{thread.thread_text}}<p>
    {% if thread.comment_set %}
        {% for comment in thread.comment_set.all%}
            <div style="border: 1px solid black"><p>   {{comment.comment_text}}</p> --- <a href='/
            users/{{comment.user.id}}'>{{comment.user.username}}</a><p>    {% if comment.user.additionaluserinfo.image %}&
            nbsp;   <img src="{{comment.user.additionaluserinfo.image.url}}" height= "70"></p>
    {% endif %}
        {% endfor %}
    {% else %}
        <p>There are no comments currently on this thread</p>
    {% endif %}
{% endif %}

{% if request.user.is_authenticated %}
<form action = 'add' method='POST'>
    {% csrf_token %}
    Comment:
    {{ form.text }}
    <input type='submit' value='Submit'>
<p>In order to comment in this thread, you must have an account. Register <a href="/register">here</a></p>
{% endif %}
{% endblock %}
```

The anchor tag <a> is also used within this, with a link embedded. Thus this text will have a hyperlink attacked of the user whom created the thread. This HTML goes on to check for a set of comments within the Thread model. and iterates through these comments in the case they are found. A tag separates this with a border separated by   (non-breaking space). The comment text is outputted with an anchor linking back to the user. An image is also imputed using an <img> tag. If there are no comments in the Thread model, the HTML outputs "There are no comments currently in this thread" embedded by <p>

Following the output of the Thread model and it's various comments within it's comment_set. A form is initialised with various necessary eSport comment creation functions. Checking first if the user is authenticated (logged in and registered) the form is then creatded with the action for adding a comment and HTTP method for "POST", as it is posting an eSports comment into the linked SQL server. A 'csrf_token' is used for protection of the data, and prevents 'Cross-site request forgery' (https://portswigger.net/web-security/csrf/tokens#:~:text=A%20CSRF%20token%20is%20a,request%20made%20by%20the%20client).
This is followed by the text submitted through the form, with an input tag for submitting the form. A paragraph informing the user to login is outputted if the user is not authenticated

Generate- Code

This following HTML file follows a similar theme to the previous, however this outputs threads and includes a thread forum for each eSport followers can use for a median of FIFA eSports discussion. The threadset (set of threads) is checked for existence and the threads within the threadset are iterated through. Through a list tag (<li>), the thread's are iterated with their title and text outputted and linked back to the user.id and the thread's own webpage. The username and the text are imbedded within these files. If no threads exist the user is informed that "No threads are available"

```
{% extends "main/header.html" %}

{% block content %}

    {% if threadset %}
        <ul>
        {% for thread in threadset %}
            <li><a href="/threads/{{ thread.id }}/">{{ thread.thread_title }}</a> || <a href='/users/{{thread.user.id}}'>
            {{thread.user.username}}</a>
            <p>{{thread.thread_text}}</p></li>

        {% endfor %}
        </ul>
    {% else %}
        <p>No threads are available.</p>
    {% endif %}


{% if request.user.is_authenticated %}
    <form action = 'addthread' method='POST'>
        {% csrf_token %}
        Create Thread <br>
        Thread Title:
        {{ form.title }} <br>
        Thread Text:
        {{ form.text }}
        <input type='submit' value='Submit'>
{% else %}
<p>In order to contribute to the threads, you must have an account. Register <a href="/register">here</a></p>
{% endif %}

{% endblock %}
```

The thread form is made through the <form> tags, with the action "addthread" and the HTTP request "POST". With a 'csrf_token' imbedded at the top, the fields are initialised with the relevant text to inform of the field's text. This is ended by a submit form button. In the case the user is not authenticated at the start of the form. Stated "In order to contribute to the threads, you must have an account, Register here" with the link for the registration embedded.

Header.html

```
{% load static %}
    <link rel="stylesheet" type="text/css" href="{% static 'main/style.css' %}">
<header>
    <nav>
        <a href="/">Home</a> |
        <a href="/users">Users</a> |
        <a href="/threads">Forum</a> |
        {% if request.user.is_authenticated %}
        <a href="/profile">Profile</a> |
        {% else %}
        <a href="/register">Register</a> |
        {% endif %}
        <a href="/lockers">Lockers</a> |
    </nav>
</header>
<body>
    {% block content %}

    {% endblock %}
</body>
```

The header.html file is integral to the styling and structure of the website, as it is the file each HTML file links to using Django's 'extend' function. A 'CSS' (Cascading style sheets) file is linked. Within a CSS file, the various colours and visual elements of the webpage are made. Through this, the basis of the visual communication elements and principles are expressed. Following this link, a header tag is made; that will thus be used within all HTML pages. A <nav> tag is used for a navigation bar, and the various links through anchor tags (<a>) are made with the name for the links and their respective links. A statement is made to check if the user is authenticated to determine whether a profile or a register tab should be showed. The {% block content %} is an integral Django function where all other HTML is embedded within, thus explaining why every HTML file excluding this one, the HTML is embedded by {% block content %} and {% endblock %}.

Evaluation and References- Impacts

Obj 5,6,7 Obj 8

Social, Personal, Economic and Legal Impacts (PC4)

Social

The Bundaberg FIFA eSports web app was designed for the purpose of the uniting the FIFA eSports community and thus the social aspect of the app is integral to it's function. Features such as the forum are designed entirely for social purposes, encouraging users to discuss FIFA related topics through thread and comments. Profiles are made for users to express themselves and interact with other user's profiles. This will be suitable for personas such as Penny who enjoys eSport discussion and competitions. Thus this site can be deemed to be socially adequate.

Personal

Within all stages of the EDGE process (explore, develop, generate and evaluate) in creation of the web app, UI and the user experience was considered. Knowledge on visual communication elements and principles attributed to the accessibility, useability and learnability of the site. The features that users were given access to also attribute to the site's appeal. With personas such as Mitch and Mandy being able to find interest in the data sets created through such feature as the forum or the data upload. Within data upload, there are algorithms and built in functions to store data in a safe way that respects the privacy of the user, using such Django functions as 'cleaned_data'.

Economic

With the rise of eSports as entertainment, it comes with larger prize money and larger potential for business ventures. Businesses increasing more often have been seen to take interest in eSports such as FIFA with the introduction of FIFA Leagues and the FIFA World Cup. Thus the Bundaberg FIFA Web App can serve as a platform for Bundaberg eSports and the business potential that comes with it. Through player profiles, data upload and team schedules, the Bundaberg FIFA Web app is adept for major business potential.

Legal

The data used within the Bundaberg FIFA Web App was open and public data, and the modules and necessary external components used to create the prototype was open source.

References

Djangoproject.com. 2020. *The Web Framework For Perfectionists With Deadlines | Django*. [online] Available at: <https://www.djangoproject.com/> [Accessed 4 June 2020].

Lucidchart.com. 2020. Lucid Chart. [online] Available at: <https://www.lucidchart.com/pages/> [Accessed 4 June 2020].

Professional forms solutions for you by Zigaform. 2020. *Portfolio - Professional Forms Solutions For You By Zigaform*. [online] Available at: <https://zigaform.com/> [Accessed 4 June 2020].

W3schools.com. 2020. *W3schools Online Web Tutorials*. [online] Available at: <https://www.w3schools.com/> [Accessed 4 June 2020].

Wufoo. 2020. *Online Form Builder With Cloud Storage Database | Wufoo*. [online] Available at: <https://www.wufoo.com/> [Accessed 4 June 2020].

Prescribed Criteria (PC) and Self-Prescribed Criteria (SC) Checklist

- Must have 7 days of FIFA related data such as player goals, assists, wins, losses, penalties (SC1) ✓
- Must have 7 days of Web app related data such as forum data (threads and comments), user details, images (SC1) ✓
- A data upload system for .csv and other files storing masses of data (SC1, PC5) ✓
- An alert system that sends notification sufficient to the magnitude of the event warranting an alert (SC6) ✓
- Clear site navigation system for easy access and learnability (PC1, PC2) ✓
- Construction of website through the use of data to gain information on the social and economic requirement and wants of the target audience for the Bundaberg FIFA eSports web app (PC8, SC2) ✓
- Efficient and learnable code that implements complex functions yet necessary for the Bundaberg FIFA eSports web app (PC7, SC3, SC4) ✓
- Allowance of users to upload data and share data such as comments, threads and images (SC1) ✓
- Visual communication elements and principles displayed within constructed of Bundaberg FIFA eSports web app UI (PC1, PC6, SC3) ✓
- Create appropriate diagrams such as DFDs, ER Diagrams and prototype UI designs to help in development stage (PC2, PC5, PC6) ✓

Constraints

- Assignment finished within time frame and considering restraints such as Coronavirus and separate outside disruptions ✓
- Assessed many accessibility standards and criteria through all stages of the EDGE process within developing the prototype Bundaberg FIFA web app (Explore, develop, generate, evaluate) ✓
- All data used and uploaded was within legal limits and was done within fair use policies ✓

Completion Criteria (To-do list)

- Create Thread and comment construction; eSports discussion forum ✓
- Use Relevant modules used for Bundaberg FIFA audience ✓
- Create Admin/ Superuser field for testing and easy access to update ✓
- Use FIFA eSport related data, fields for input and data upload ✓
- Use the Django-exclusive features that minimise development ✓
- Create UI's considering accessibility, useability and learnability ✓
- Create relevant fields for user registration and profiles such as 'username' and 'password' ✓
- Generate code to register and store data inputted by users through generated forms ✓
- Create an alert system for users with varying settings ✓
- Create method for the encryption and decryption of data to respect privacy of users ✓

Evaluation- Testing and Recommendations

Event/ Action	Theoretical Output	Actual Output	Improvements/ Extensions
Test inserting models such as threads, comments and users	Models should be inserted into SQL table and should be outputted as normally	Worked as expected	
Test registering a user	All fields such as username, first_name, last_name and password should be inputted into the SQL table along with password being encrypted	Worked as expected, with the addition of Django's auto-generated 'id' column	The password encryption could be used from an external source
Test adding a thread via the ThreadForm	The fields should be transferred via the python code to the SQL table and the page should refresh with the new table	Worked as expected	
Test uploading images	The Image should be filed into a datastore and be rendered within HTML code to display to the front-end	Worked as expected	Unreliable as only a flat-file database used for this. A larger more reliable database is needed
Test attempting to reach admin controls with a non-admin account	The user should be denied of access	Worked as expected, within the Django admin panel	A more customised admin panel could be implemented, more suited to the FIFA Web app
Test the hyperlinks and various URLs	The links direct the user to the correct page, with the correct text and titles displaying links	Worked as expected	Make links more clear as a link, blue underline is often universally accepted as a link
Test adding a comment via the CommentForm	Similar to the ThreadForm, the fields should be transferred via python to the SQL table, the page should refresh with the comments rendered	Worked as expected	
Test inserting a thread with no fields	The thread is not added	Thread was added with autogenerated 'id' and null for all nullable fields	Implement a requirement for one or more fields to be NON-NULLABLE in the Thread Model

Obj 5,6,7

Accessibility Checklist

- A clear navigation bar that gives access to tabs such as 'Users', 'Home' and 'Threads'
- Forms are correctly aligned for greater useability and learnability
- Clear borders used for objects such as threads and comments
- Colour contrast between text and background
- Links describe the URL location accurately and efficiently
- Colour not used to display information (body text is not coloured)
- Text size can be manipulated and augmented by the user
- Fields that are mandatory are indicated to be so (Fields that are 'NON NULLABLE' within the SQL Table)
- Users can customise their profile with a profile image
- Breadcrumbs are implemented to assist with navigation for the useability and learnability of the site
- Multiple files can be uploaded such as .csv files
- Labels are positioned close to the information related
- There is equal proximity between most/ all objects within any given web page
- URLs are readable, easy to interpret and navigate
- Alerts are readable and have adequate settings for the magnitude of the error
- The Bundaberg FIFA eSports Web App main colours can be seen throughout all the app's UI components

Further Recommendations

Despite the completion of the prototype, it is evident that there is potential for many of the functionalities and aspects to be improved or extended upon. Although the prototype showed to have many integral features within many web apps such as users, registration/login and a forum, there is much eSport and FIFA related content that can yet be added.

An eSport feature that is common within eSport hubs online is a streaming feature. Using modules from 'Twitch' or other streaming platforms would attribute to the eSport experience. This could be added upon with a live chat for users to collaborate in the midst of an eSport event. Whilst explored, the app never solidified the concept of adding eSport 'team pages' for FIFA Professional eSport gaming teams. Team pages can be customisable and much access can be given to other team developers. Schedules can be implemented for the competition along with ladders and tables with live scores. The implementation of these eSport companies and teams would attribute greatly to the Economic factor of the Bundaberg FIFA Web app.

There are also many FIFA specific components that can be added. FIFA, being a constantly updating and changing game with data such as player statistics and league expectations being changed regularly, there could be more standardised output of this data within the Bundaberg FIFA Web app. As opposed to leaving this discussion to the forum, official pages such as 'FIFA Updates' could be implemented.

More connection to Bundaberg was needed. This could be done by locating an area for meeting development and maintenance of this site in Bundaberg along with a map located on the home page.

Obj 1,2 Obj 3,4 Obj 5,6,7 Obj 8

Digital Solutions

Bundaberg eSports FIFA Web App
IA2 Project; Prototype

Paddy Maher

0:00:04 0:03:47