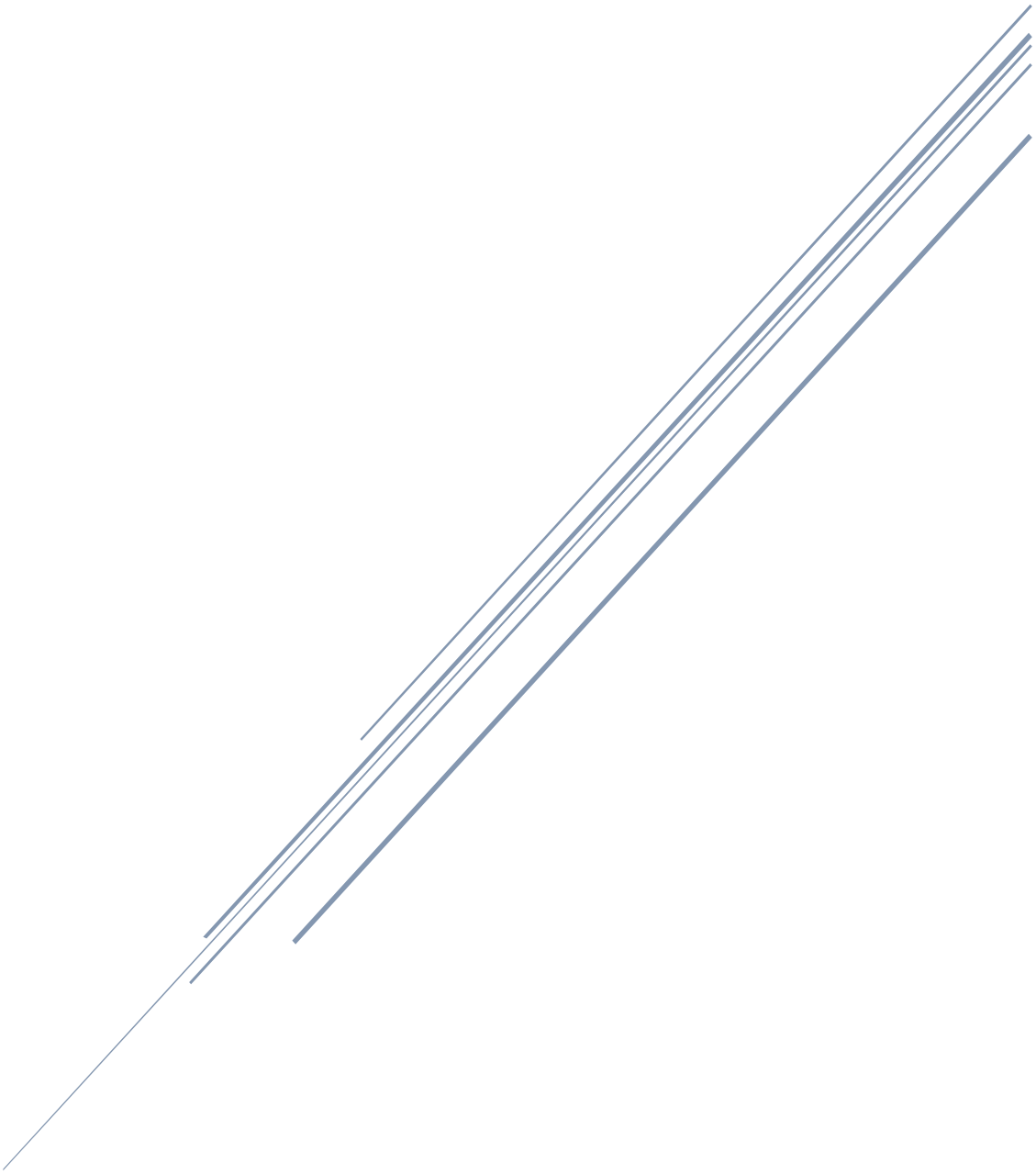
NEA

By Kamran Bhatti



Harris Academy South Norwood
Computer Science

Table of Contents

ANALYSIS

Problem Background

I believe that the exposure of children to profanity in social media has been the driving force for them to develop aggressive tendencies and has caused academic hindrances.

In my experience, students who use social media devote a large portion of their time digesting the latest trends or gossip that was circulating at the time due to having all of this information accessible from the click of a button. The allure of social media is to allow people to stay in touch with peers, all the while, students become consumed by their need to maintain their digital presence rather than how they choose to be represented in real life as shown by a lack of interest in their own education as it is not as intriguing or emotionally stimulating.

In modern society students are, on occasion, not monitored when on their digital devices, thus causing students to give into temptation and peruse the internet for the latest gossip causing their studies to be neglected. This correlation leads to students who frequent social media to attain lower grades than their counterparts causing some students to become antisocial coupled with a lack of communication skills which deprive students of future opportunities to develop their academic careers.

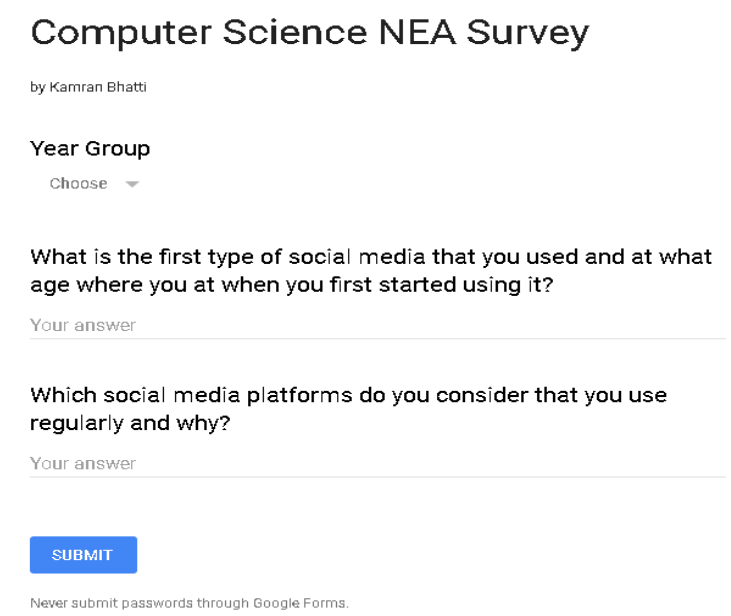
Additionally, students may encounter forms of cyber bullying which could cultivate self-harming, having suicidal thoughts or isolating themselves both in reality and virtually. While being led to believe that they are ostracised by everyone around them, these individuals may adopt the notion that they are unable to access avenues which could provide them the guidance or help they need. However, the greatest threat these students face is their own naivety; from succumbing to peer pressure by posting unfavourable content to being coerced into sharing explicit images. Whilst not understanding the detrimental ramifications of their actions, any future opportunities to seek employment or further their academic career to their privacy online could all be relinquished as they do not realise that what is on the internet leaves a trail that once posted can’t be expunged.

I propose that it would be beneficial for schools to endorse students using a platform that the institution can monitor which will eventually lead to students being able to maintain both a virtual presence and academic career. In turn, this will help alleviate the threats of normal social media and will nurture students in a safe online environment.

Designing and Distribution of Surveys

Firstly I wished to gather data from a range of people from my target age group, which I decided would consist of Year Seven to Upper Sixth Form; ages at which students are generally introduced to social media or frequent the use of social media within their daily lives. In order to gather the data I required I distributed surveys to Year 7, 8, 9, 10, 11 and the Sixth Form at Harris Academy South Norwood. An example survey is shown below:

Questionnaire 1 in Google forms



Computer Science NEA Survey

by Kamran Bhatti

Year Group

Choose

What is the first type of social media that you used and at what age where you at when you first started using it?

Your answer

Which social media platforms do you consider that you use regularly and why?

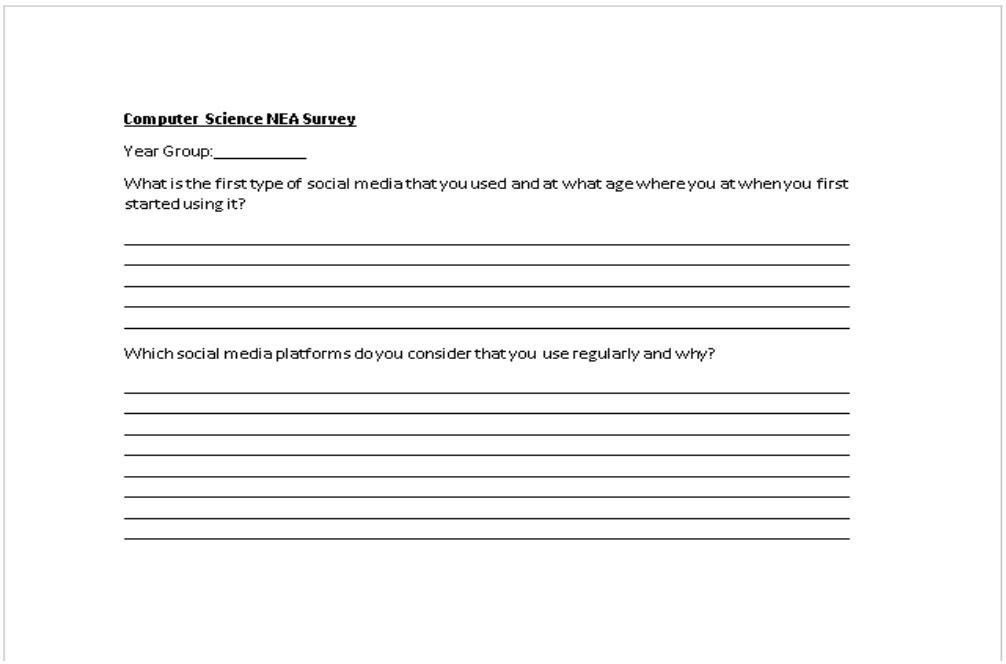
Your answer

SUBMIT

Never submit passwords through Google Forms.

https://goo.gl/forms/QifKvKvNWHI9oj7p2

Questionnaire 1 in a word document



Computer Science NEA Survey

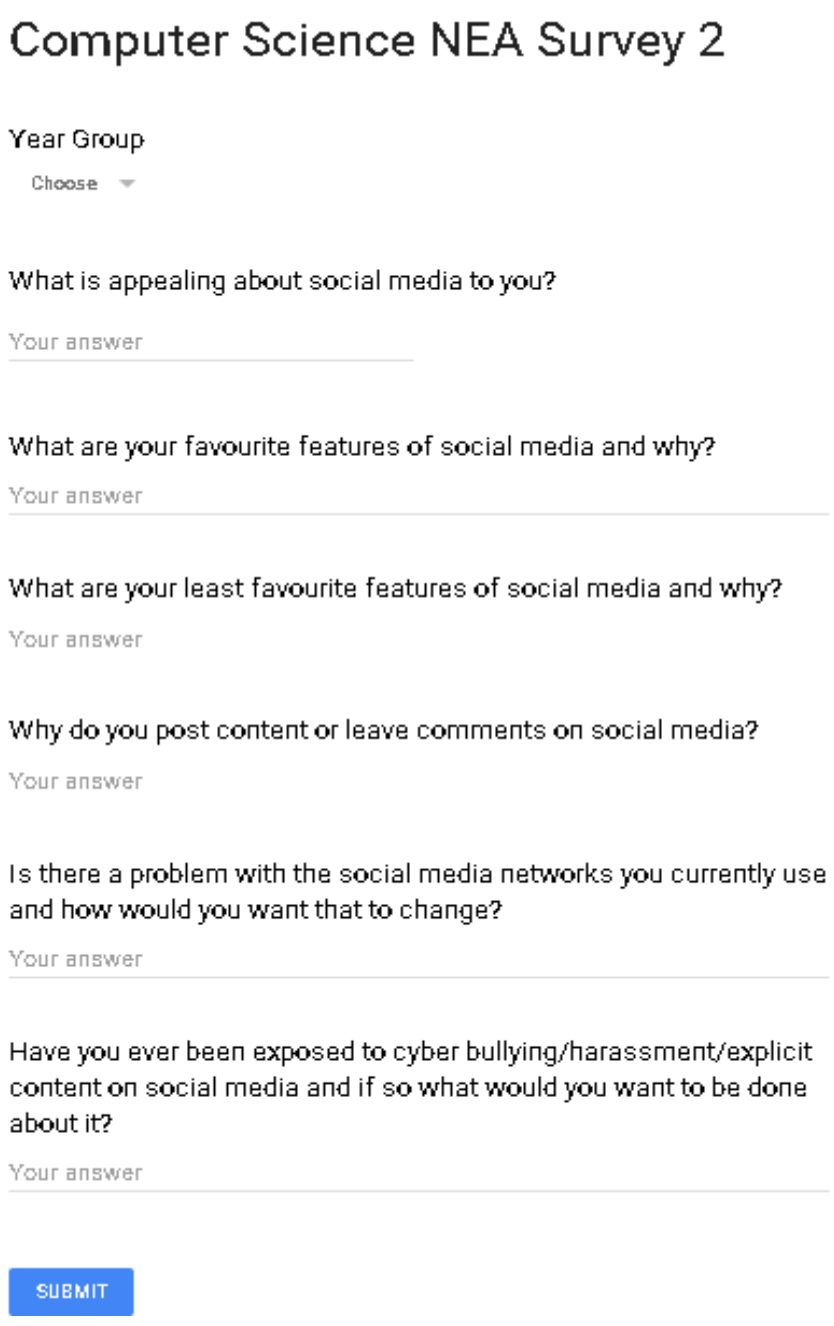
Year Group:_____

What is the first type of social media that you used and at what age where you at when you first started using it?

Which social media platforms do you consider that you use regularly and why?

I phrased my second question in a way that would make participants consider what they “use regularly” as a form of social media seeing as my targeted audience for the survey predominantly uses multiple instances of social media in nearly every aspect of their day. The notion of “regularly” would make participants delve into discerning which platforms and types of platforms they frequent and the appeal these platforms have.

Questionnaire 2 in Google forms



Computer Science NEA Survey 2

Year Group

Choose

What is appealing about social media to you?

Your answer

What are your favourite features of social media and why?

Your answer

What are your least favourite features of social media and why?

Your answer

Why do you post content or leave comments on social media?

Your answer

Is there a problem with the social media networks you currently use and how would you want that to change?

Your answer

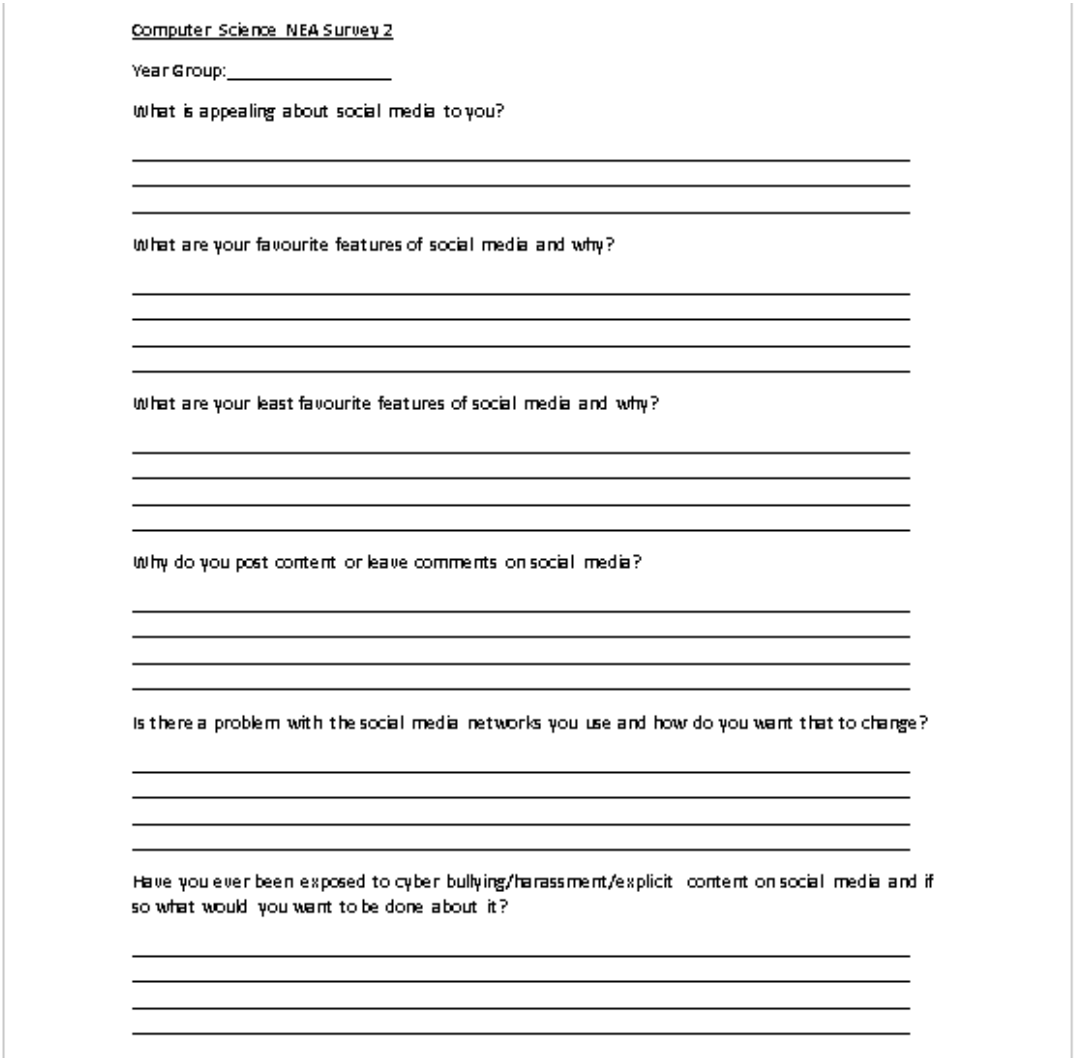
Have you ever been exposed to cyber bullying/harassment/explicit content on social media and if so what would you want to be done about it?

Your answer

SUBMIT

https://goo.gl/forms/HjwMVsgAOJ6ZDaHz1

Questionnaire 2 in a word document



Computer Science NEA Survey 2

Year Group:_____

What is appealing about social media to you?

What are your favourite features of social media and why?

What are your least favourite features of social media and why?

Why do you post content or leave comments on social media?

Is there a problem with the social media networks you use and how do you want that to change?

Have you ever been exposed to cyber bullying/harassment/explicit content on social media and if so what would you want to be done about it?

Using a multitude of data collection methods from surveys and interviews to online resources, I was able to scrutinise any opinionated sections where I discussed the problems background against realistic figures as depicted below.

Interviews

In order to test the adverse effects of social media in a school setting I conducted interviews with a boy and girl from each year group, from Year 7 to Year 13 as most of the research I conducted into the background of the problem had found that most figures for the effects of social media had effected age ranges between 11 and 18 years old.

Amongst this group of individuals, all participates had agreed that they have used a form of social media either recently or in the past week.

All interviewees quickly agreed that they had experienced a form of harassment on social media in the past but when asked about aggression towards being harassed, 5 of the 7 boy's and 3 of the 7 girl's answers were generally similar in stating:

"It is annoying to the point where I feel like my head is going to explode."

Whereas the remaining answered similarly in saying that:

"It is bothersome but it's just words. They are just people trying to get a reaction."

Although the last statement depicts how there are students who do not display any aggression towards being harassed as they feel it is just a ploy instilled to evoke a reaction, here it's evident that all participants had experienced harassment in one form or another. However, this also indicates that students feel a form of aggression towards being harassed which may have a greater influence on boys considering the majority of them responded emotively in stating that they "feel like [their] head is going to explode" which is also shared among some of the girls showing that they also adopt this form of aggression.

Furthermore, I proceeded to ask how long students tend to spend on social media during the week. Interviewees responded differently depending on Year group.

Students from Year 11 and 13 answered with 2-5 hours per week.

Year 7 and 8 students answered with 12-15 hours per week.

Year 9 and 10 students answered with 10 hours per week.

Year 12 students answered with 5-7 hours per week.

This seems relatively similar to my expectations as I expect students in Year 11 and 13 to have less free time due to exam preparation and work load. Year 12 have more free time than Year 11 or 13 as they don't have to prepare for examinations thus allowing them time to peruse the internet at their leisure in their free time. Surprisingly, students in Year 7 to 10 spend more than 9 hours per week using social media which could supplement as to why students are attaining low grades due to perusing the internet for a large amount of time to which they neglect their studies.

I proceeded to ask if the students had ever experienced cyber bullying.

Students in Year 7 and 8 quickly answered by saying:

"No I haven't"

From Year 9 to 13 there were mixed responses. 3of the 5 remaining boys stated that they had experienced cyberbullying whereas 2 of the 5 remaining girls agreed in stating:

"I have in the past but not anymore"

Whereas the remaining participants shared the views of the Year 7 and 8's. It was surprising to see that no Year 7 or 8 students had experienced cyber bullying however, it is shown by my findings that cyberbullying although not prevalent, still exists in a school setting as depicted by 5 students experiencing a form of cyber bullying while attending the institution. As my findings are coming from a small sample of the school this number may be skewed in which more students may be experiencing cyberbullying than depicted within my findings which further supports the need for an establishment like a school having a social media platform where they can monitor cyber bullying.

Lastly, I asked if any of the interviewees had sent or received explicit content through social media.

The Year 7 to 9 students stated that they hadn't received or sent explicit content through social media.

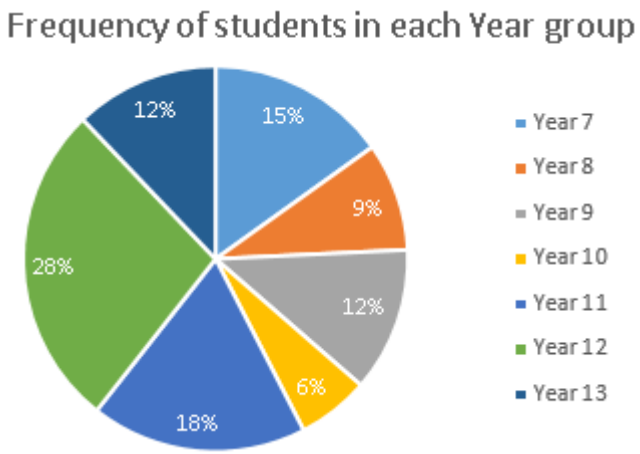
From Year 10 to 13 I received a mixture of responses. Half remaining boys said they had sent explicit content through social media and half said they received explicit content through social media. 1 of the remaining girls said they had sent explicit content through social media however, 3 of the 4 girls said they had received explicit content through social media. This suggests that the higher Year groups within the institution have been exposed to explicit content through social media or have shared explicit content through social media.

Overall, I believe that these interviews supplement to how detrimental social media can be to students in an educational institution in consideration to how students are exposed to cyberbullying, harassment and explicit content to how students frequent social media to the point where their academic performance is severely neglected. This shows that unmonitored exposure to social media can be detrimental to a student's academic development. I believe that there is room to improve the current application of social media to incorporate techniques to improve safe guarding on the platform. This could reduce exposure to cyberbullying or explicit content which could also encourage students to devote more time to their education and studies outside of school as this age group uses social media to the point where they experience academic deficits.

Feedback

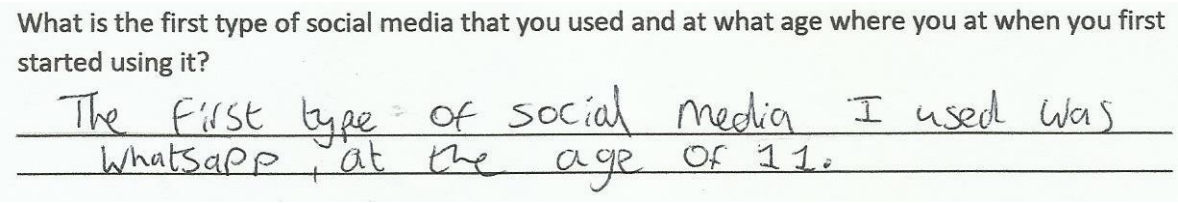
For some of the questionnaires I converted the data collected to pie chart form as to show the data in a way that could easily be understood.

Year Group



This shows that of the 33 students that I interviewed and had given questionnaires to, the majority of students who participated were in Year 7, 9, 11, 12 and 13.

The students from Year 12 stated that they started using social media from as early as 5 with platforms like YouTube to the age of 14 with platforms like WhatsApp, Facebook and Snapchat as shown below:


What is the first type of social media that you used and at what age where you at when you first started using it?

The first type of social media I used was Whatsapp, at the age of 11.

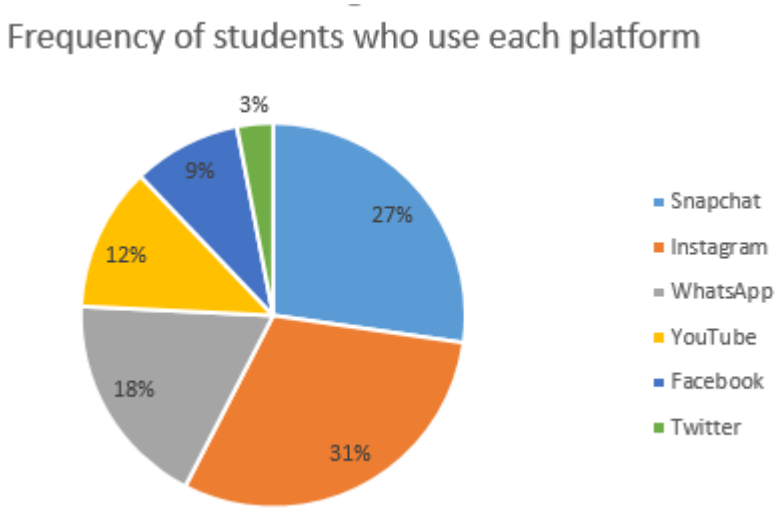
For Year 13 students, it was conclusive that these students started using social media as early as 12 or 13 with Facebook.

The data retrieved from Year 7 to Year 11 student however were very interesting considering that the Year 11 students stated that they had started using social media from the age of 12 but with the Year 9 and 10 students stating that they started using social media at the age of 10 and that the Years lower started using social media platforms from the age of 7.

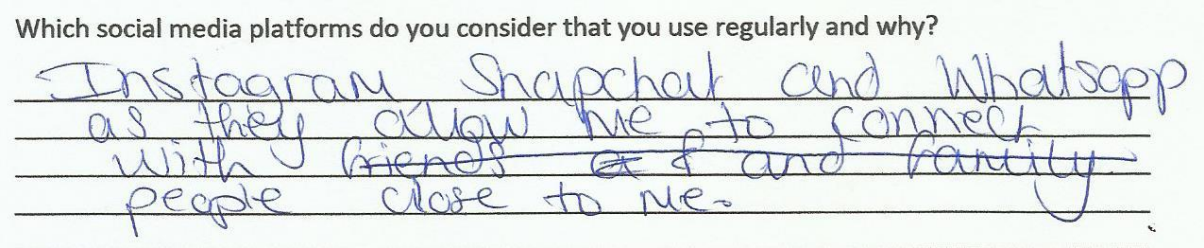
I believe that this clearly shows how the advancement of technology has allowed students to be exposed to social networks from a starting age which depreciates as the years go on.

When asked what networks students use regularly and what seems to be appealing about these networks, the results were skewed towards Snapchat and Instagram as shown below:

Number of students using each platform



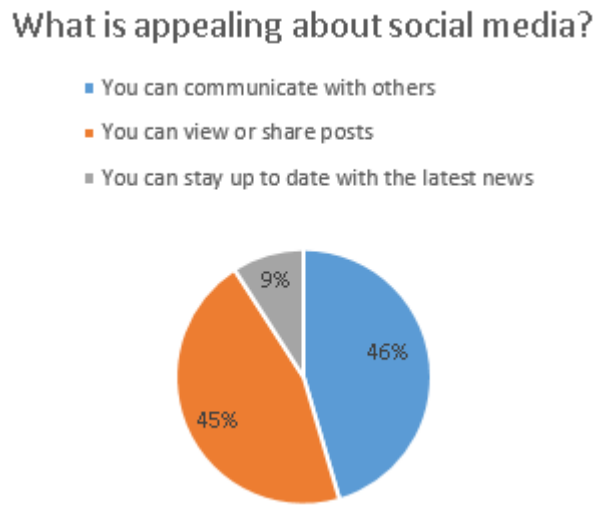
This depicts that most of the 33 students who participated are using Snapchat or Instagram compared to other social media platforms. When questioned on why students use these platforms the students responded by saying that they used the platform to interact and stay in touch with friends and family members however, the spread of data shows that most students use other forms of social media as well as Snapchat and Instagram. This is depicted below:

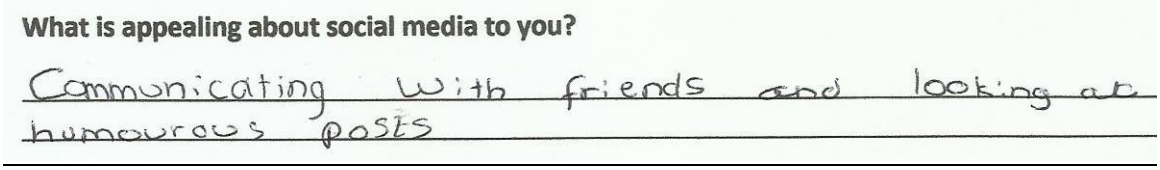


However, the only platforms which are suitable for this age group being in secondary school in YouTube and WhatsApp and considering that the frequency of students using social media platforms other than these is quite large show that there is a need for a system to be instilled to monitor the content that students have access to on social media as although the age requirement for these platforms are clearly stated when signing up, students disregard these warnings and continue to use the platform anyway.

With the same group of students I continued to hand out my second questionnaire.

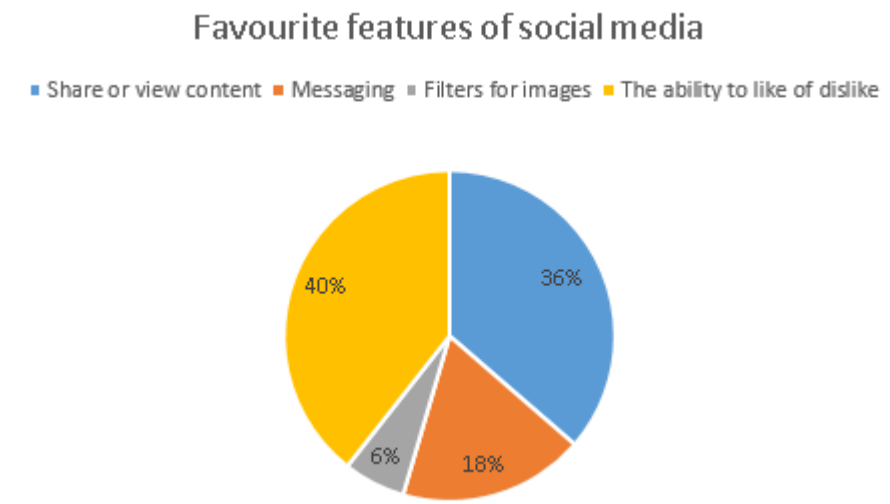
What is appealing about social media?

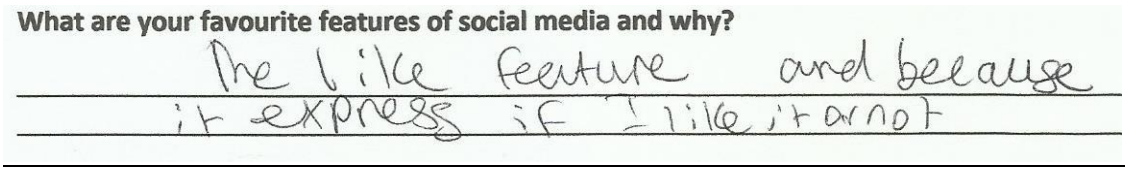




As shown most responses align with the idea that you can communicate with others and being able to view of share posts while a small majority use the social network to stay up to date with current affairs.

What are your favourite features of social media?

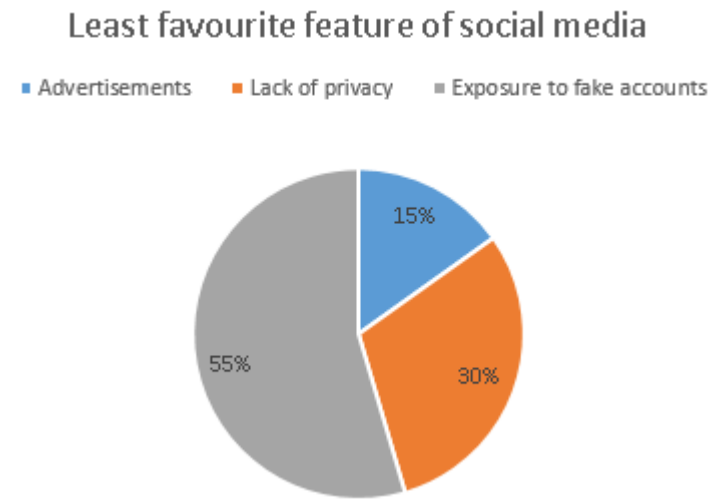




This depicts that the majority of students use social media with the intent to share or view content and because they are able to express if they like or dislike content on the platform so that the

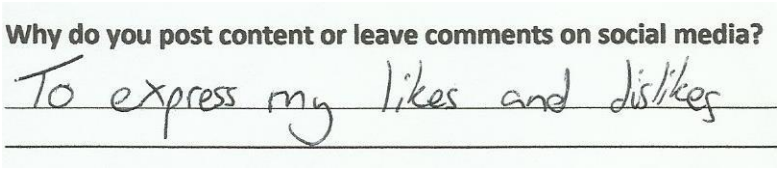
content creator is able to supply more content to the user. However, a considerable amount of users use social media to message others and to create filters when taking images.

What are your least favourite features of social media?

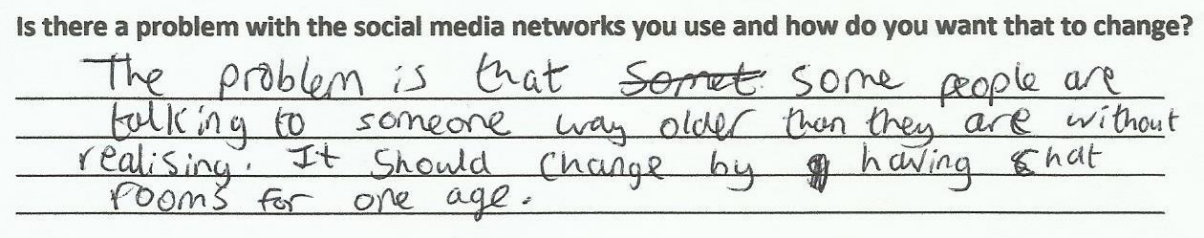


This depicts that a considerable number of students find that advertisements are their least likeable feature of social media and when asked why, students tended to state that it takes away from the experience when interacting with the social media platforms. However, surprisingly when considering the majority response was the lack of privacy and exposure to fake accounts; when asked why students found this feature to be least favourable the response was very similar in stating that you could be exposed to dangerous people or that you can be exposed to abusive or slanderous content.

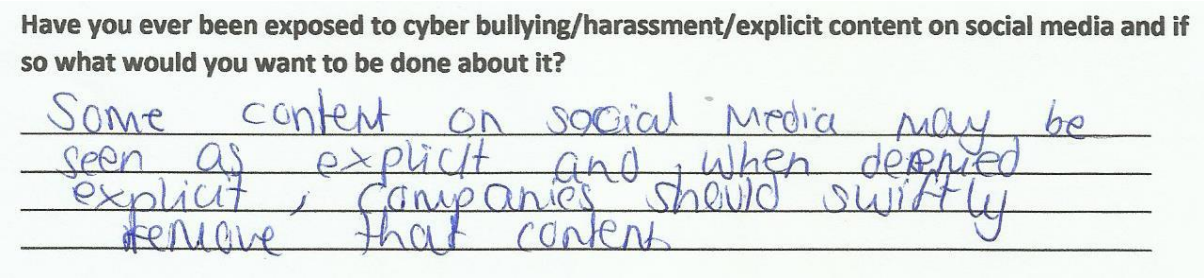
When asked why students post content or leave comments of social media the response was very similar in stating that they want to be able to share their opinion on a post or so that they are able to express themselves by sharing what they are doing with others.



Additionally, when asked if there is a problem with social media networks that the students use and how they want them to change, I received a variety of vague responses which tend to align with a single commonality. Either the service they are using is burdened with needless features which renders the service unfavourable such as being exposed to advertisements when on the service or, students stated that they felt that cyberbullying and exposure to predators on the internet is the main problem as these individuals are able to hide behind the anonymity of the network. When asked how they want this to change students stated that they want more precautions put into place to ensure the safety of the users on the platform and that needless features should be removed from the platform to ensure that the platform is user friendly.



Finally, when asked if students had been exposed to either cyber bullying, harassment or explicit content through social media and what they would want to change to ensure that it doesn't happen in the future, the responses I received were very similar. Students responded by saying that when dealing with cyberbullying, harassment or explicit content, they want the content to be disposed of swiftly and that they want the administrators of the network to get into contact with the individuals who are affected.



Conclusions

From the initial survey I can conclude that my target audience for my project should be for students in secondary school. This is evidenced by students in secondary school stating that they started using platforms like Snapchat, Facebook and Instagram from as young as 12 which is significantly younger than the age that is specified on the terms of service for users of the platform. Furthermore, as students are using services from such as young age I will be creating a project which aims to improve safe guarding and reduce exposure of young adolescents to inappropriate content via social media.

From the second survey use social media platforms to communicate with others and to share or view posts. I can also infer that students want to use a platform which has some form of like or dislike feature as it allows users to convey to content creators or posters their preferences on what they decide to share with others. Furthermore, I also acknowledge that advertisements on social media networks can be intrusive and unpleasant when using the platform for long sessions however, the greater issue which was raised was the exposure to fake accounts and lack of privacy. This further reinforces my proposal to create a platform which ensure the safety of students online when on social media.

Therefore, I will be incorporating like or dislike features into my platform as well as implementing a system to block and report users of the service as to ensure the safety of users and to reduce the exposure of students to cyberbullying, harassment and explicit or vulgar content. Additionally, I will implement an administrative role for teachers to be able to use so that teachers are able to view the posts and comments of users in means of deleting the respective comments or posts or deleting accounts from the social network if they choose to.

What does the student need?

To be able to interact with the social network the student will need me to allow them to sign up and login to the network. The student will also need to be able to complete their user profile and view

their own profile. They will also need to be able to create posts, view their own and other people's posts, delete their own posts, like their and other peoples posts, comment on their and other peoples posts and to be able to search for posts. The user will also need to be able to add friends, view their friends list, view other users' profiles, and remove their own friends and to search for friends to add.

What does the admin need?

The administrator needs to be able to view all the users of the social network, delete users of the network, delete posts of each user, delete comments of each user and view what users are friends. These actions that the admin can carry out should not be accessible to normal users, only users with super user permissions should be able to access the admin side of the social network.

Additional research into the problem

- **"the exposure of children to profanity in social media has [caused] them to develop aggressive tendencies"**

A paper titled "Online Aggression: The Influences of Anonymity and Social Modelling"[1] written by Adam G. Zimmerman under the University of North Florida. It states that the increase in inappropriate behaviour online (which is referred to as cyber disinhibition) occurs mainly due to the anonymous nature of the internet:

"Individuals may behave in ways that contradict normative behaviour [as they] are able to freely make any statements [or] be whoever they want to be [and simply] log off at the end of the day. This ability to disconnect might trump the need for permission to behave in certain ways. This [ability] to disengage with the click of a mouse might lead one to behave drastically different in comparison to [a face to face] interaction where this radical sovereignty does not exist."

I believe that this is a vital point; students may partake in cyberbullying or developing an aggressive behaviour as they are not limited by the constraints that are faced in the real world such as the repercussions of their actions. The notion of anonymity allows students to behave in a way that 'contradicts normative behaviour' as they are able to disengage emotionally from their actions.

According to an article on the Daily Cougar titled "Social media encourages antisocial, aggressive behaviour" written in July of 2016[2]; it is stated that 78% of users reported a rise in hostility on social media and 2 in 5 people blocking or unfriending someone due to a virtual dispute which was founded in a study by Joseph Grenny and Forbes contributor in 2013 to which:

"In May [2016,] 18-year-old Kayla VanWert and 16-year-old Cathleen Boyer, started a feud on Facebook. The two arranged to meet in an alley where Boyer fatally stabbed VanWert in the neck."

This further reinforces the notion of students nurturing aggressive tendencies within their behaviour as the exposure to cyberbullies or harassment could lead to real life repercussions.

- **"[The use of social media] has caused academic hindrances."**
- **"[Students] are not monitored when on their digital devices, thus causing students to give into temptation and peruse the internet for the latest gossip causing their studies to be neglected."**

[1] https://digitalcommons.unf.edu/cgi/viewcontent.cgi?article=1472&context=etd
[2] http://thedailycougar.com/2016/07/27/social-media-encourages-antisocial-aggressive-behavior/

In a research paper titled "Effect of Social Media on Academic Performance of Students in Ghanaian Universities: A Case Study of University of Ghana, Legon" written by Kolan John Bernard and Patience Emefa Dzandza under the University of Nebraska – Lincoln[3], it is stated that 31.5% of students responded that they didn't experience any improvement in their grades.

Furthermore, a paper titled "Influence of Social Media on the Academic Performance of the Undergraduate Students of Kogi State University, Anyigba, Nigeria" written by Ezekiel S. Asemah and Ruth A. Okpanachi from Kogi State University and Leo O.N. Edegoh from Anambra State University[4] depicts the negative influence of the academic performance of undergraduate students of Kogi State University students as to show that students who spend more time on social media are likely to have inadequate performance academically.

"Table 4".

S/N	ITEM	SA	A	UD	D	SD	X	DECISION
1.	Kogi State University undergraduate students spend more time on social media than reading their books	60	199	10	2	0	4.1	Accepted
2.	The students now rely on social media to do their assignments without consulting other sources	71	181	10	7	2	4.1	Accepted
3.	Students' exposure to social media have effect on their academic performance	107	149	7	4	4	4.2	Accepted
4.	The influence of social media on the academic performance of students is negative	13	104	27	0	2	4.3	Accepted
5.	Students who spend more time on social media are likely to perform poorly in their academic activities than those who do not.	77	177	17	0	0	4.2	Accepted

This instils the notion that students are using social media to the point where their performance academically is affected adversely as the exposure of undergraduate students at the institution to social media is high and has a negative effect on their performance.

Oberiri Destiny Apuke states in a research article titled "The Influence of Social Media on Academic Performance of Undergraduate Students of Taraba State University, Jalingo, Nigeria"[5] that of respondents; 75% strongly agree that the influence of social media on academic performance of students is negative, 10% agree that the influence of social media on academic performance is negative whereas, 10% disagreed with the notion with an additional 5% strongly disagreeing.

"This implies that to a very large extent [that] social media [has a] negative influence on [students'] academic performance."

These findings suggest that students are influenced heavily by social media. As well as reinforcing to me that an educational application of social media may be effective, I believe this expresses how students would benefit in using social media. Many 'respondents' consider that there are adverse

[3] https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=4687&context=libphilprac
[4] https://pdfs.semanticscholar.org/2b7f/1561d3fed7fe48ec5c88281f7d7d9b74e70d.pdf
[5] https://www.researchgate.net/publication/317083347_The_Influence_of_Social_Media_on_Academic_Performance_of_Undergraduate_Students_of_Taraba_State_University_Jalingo_Nigeria

effects to using social media when in regard to academic performance however, if the social media network was developed with educational prospects in mind it may allow students to spend time equally on digital and educational aspects of their lives.

- **"Students who use social media devote a large portion of their time digesting the latest trends or gossip that was circulating at the time due to having all of this information accessible from the click of a button."**

In a clinical report titled "The Impact of Social Media on Children, Adolescents, and Families" written by Gwenn Schurgin O'Keeffe, MD, Kathleen Clarke-Pearson, MD, and COUNCIL ON COMMUNICATIONS AND MEDIA under the American Academy of Paediatrics[6] it is stated that 22% of teenagers log on to their favourite social media more than 10 times a day and more than half of adolescents log onto a social media site more than once a day. This shows how adolescents are addicted or obsessed with social media.

In an inquiry report titled "Safety Net: Cyberbullying's impact on young people's mental health" written by Alex Chalk, Sarah Brennan and Matthew Reed[7], a survey was conducted which found that 44% of children and young people spend more than 3 hours per day on social media whilst 1 in 10 reported always using social media overnight between midnight and 6 am.

- **"[Students] may encounter forms of cyber bullying which could cultivate self-harming, having suicidal thoughts or isolating themselves both in reality and virtually."**

Written in an article titled "Concerns Regarding Social Media and Health Issues in Adolescents and Young Adults" by the American College of Obstetricians and Gynaecologists[8], 20-40% of adolescents report having been victims of cyberbullying which cause increased anxiety, depression and low self-esteem. In conjunction with physical effects like stomach-aches, sleep problems, headaches, tension, bedwetting, fatigue and a poor appetite, an analysis of 33 studies concluded that victims of bullying are at risk to lower grades.

"More recent studies have shown that bullying is associated with increased substance use, violent behaviour, unsafe sexual behaviour, suicidal behaviour, and likelihood to carry a weapon."

In an article titled "Social media: Cyberbullying" written by the Department of Health from the Government of Western Australia[9], it is depicted that every 7 seconds someone is cyberbullied with approximately 50% of victims knowing the perpetrator and approximately 50% of victims meeting the perpetrator online and didn't know them in person.

In a newsletter titled "Social Media Dangers and its Impact on Cyberbullying" from the MMA (Mahwah Municipal Alliance)[10], it is proposed that; 83% of victims felt that bullying hurt their self-esteem with 30% of children who had been bullied had suicidal thoughts and 10% of children

[6] https://www.landspitali.is/library/Sameiginlegar-skrar/Gagnasafn/Klinisk-svid-og-deildir/Kvenna--og-barnasvid/Gogn-fyrir-radstefnur/Hinn-gullni-medalvegur-Fyrirlestrar-2017/Clinical-Report_The-Impact-of-Social-Media-on-Children-Adolescents-and-Families.pdf

[7] https://www.childrenssociety.org.uk/sites/default/files/social-media-cyberbullying-inquiry-full-report_0.pdf

[8] https://www.acog.org/Clinical-Guidance-and-Publications/Committee-Opinions/Committee-on-Adolescent-Health-Care/Concerns-Regarding-Social-Media-and-Health-Issues-in-Adolescents-and-Young-Adults

[9] https://gdhr.wa.gov.au/-/cyberworld-cyber-bullying

[10] https://static1.squarespace.com/static/55aeb103e4b03ff1478dc09f/t/58b6f382e6f2e16000428b9f/1488384903033/CyberSafetyNewsletterMMAFINAL.pdf

attempted to take their own lives due to bullying. 19% of cyberbullying is due to the spreading of rumours with 72% of children report that they are cyberbullied because of how they look and 26% of victims are chosen due to their race or religion. 87% of adolescents have witnessed cyberbullying with 24% of children not knowing what to do when they are harassed and 39% of children not enabling privacy settings on social media. This emphasises how dangerous social media can be to a young adolescents' development if not in a safe and nurturing online environment.

I believe that this is in imperative reason as to why there should be a platform of social media which is monitored by an educational institution. Many students are exposed to cyber bullying which cultivates feelings of aggression, anxiety and hurts students' self-esteem. With an application of social media that is monitored by a school, cyberbullying can be prevented and actions can be taken to help students who need assistance or guidance when dealing with bullying or exposure to harassment.

From an article in the Guardian titled "Is social media bad for young people's mental health?"[11] It is proposed in a McAfee poll in 2014 of 11 to 17 year olds that 35% reported that they experienced cyberbullying which is up from 16% the year before. It is also proposed that another organisation found that Google searches for cyberbullying surge at the start of the academic year.

It is stated on an article on Bullying UK under "Effects of cyber bullying"[12] that 20% of children and young people indicate fear of cyberbullies made them reluctant to go to school with 5% self-harming, 3% attempting suicide as a direct result of cyberbullying. 28% of young people have reported incidents of cyberbullying on Twitter with 26% of young people reporting incidents on Ask.fm

In a research paper titled "Associations between social media and cyberbullying: a review of the literature" written under the University of California Institute for Prediction Technology in 2016 by Renee Garett, Lynwood R. Lord and Sean D. Young[13] it is stated that the majority of students aged 12 to 18 in the United States reported that they were cyberbullied at least twice during that past year.

- **"[Students] may adopt the notion that they are unable to access avenues which could provide them the guidance or help they need."**
- **"[The] greatest threat these students face is their own naivety; from succumbing to peer pressure by posting unfavourable content to being coerced into sharing explicit images."**

In an article written by Kelly Wallace titled "Chances are, your teen has sexted" from CNN[14], it states that more than half the undergraduate students who participated in an anonymous survey said they sexted when they were teenagers with nearly 30% including photos in their sexual text messages and 61% didn't know that sending nude photos via text could be considered child pornography. 92% of teens who said they weren't pressured to sext reported no problems afterward but that is only 68% for teens who felt pressured into doing it. This indicates that a significant proportion of participants had succumbed to naivety and pressure thus resulting in them not being fully aware of the repercussions of their actions.

[11] https://www.theguardian.com/mental-health-research-matters/2017/jan/20/is-social-media-bad-for-young-peoples-mental-health
[12] https://www.bullying.co.uk/cyberbullying/effects-of-cyberbullying/
[13] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5344141/
[14] https://edition.cnn.com/2014/11/18/living/teens-sexting-what-parents-can-do/index.html

In a report prepared for the NSPCC titled "A qualitative study of children, young people and 'sexting'" written by Jessica Ringrose (Institute of Education, London), Rosalind Gill (King's College, London), Sonia Livingstone (London School of Economics) and Laura Harvey (Open University)[15], it is proposed that 12% of 11-16 year olds in the UK have seen or received sexual messages online with 2% receiving them more than once a week compared to 15% across Europe. 11-12 year olds are less likely to receive sexual messages online than 15-16 year olds, 5% and 20% respectively. 4% of 11-16 year olds say they have posted or sent sexual messages in the past 12 months. Of 12% who had received some type of sexual message with one quarter who reported being upset or bothered by this with 4 in 10 people blocking the person who sent the message and 4 in 10 people not telling anyone that they were bothered by the experience. In a survey among 11-18 year olds, half knew of cases where sexual messages had been circulated beyond the original recipient and 30% knew someone who has been adversely affected by sexting. From a nationally representative of US mobile phone owners aged 12-17, 16% had received a sexually suggestive nude or nearly nude photo or video of someone they know.

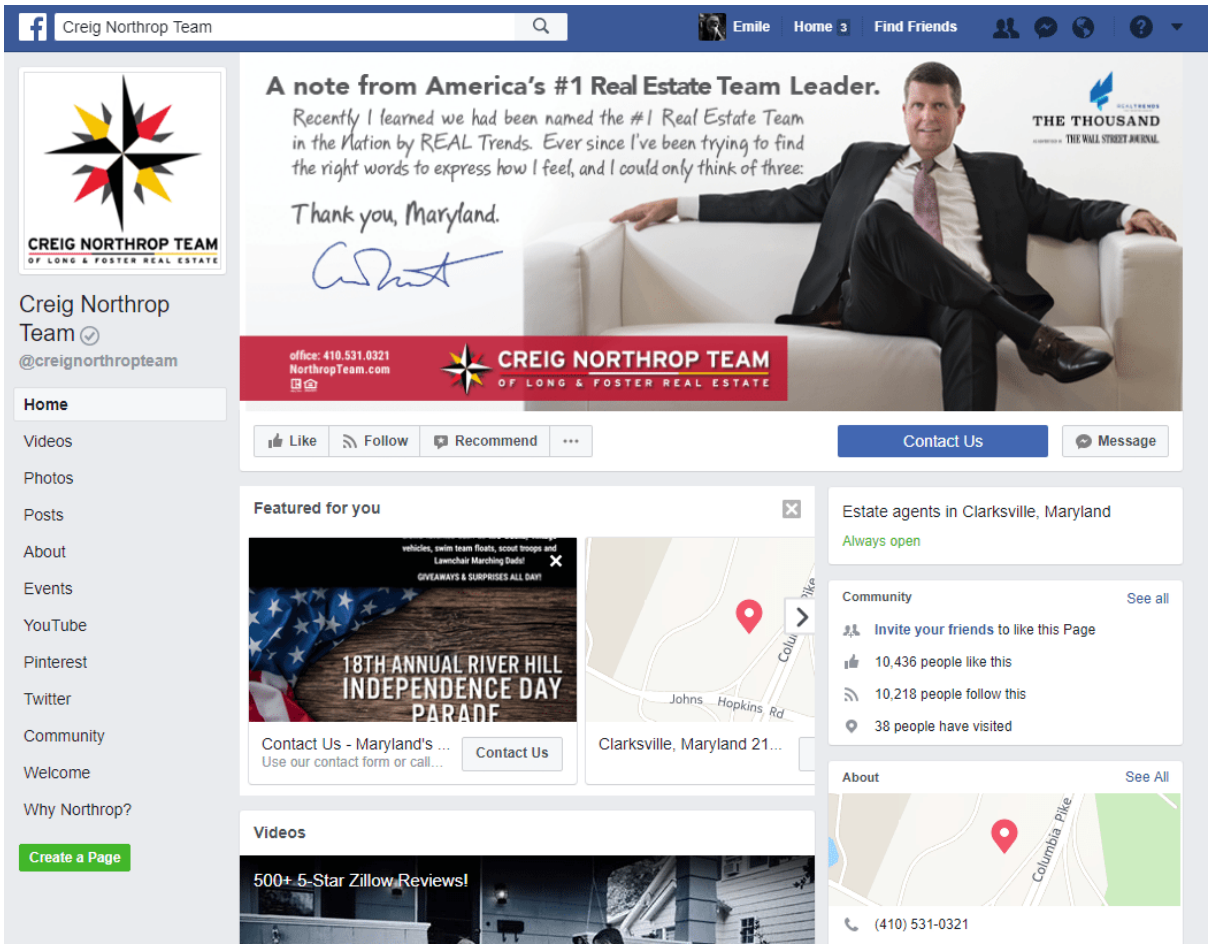
This reaffirms how students should be monitored when using forms of social media. This displays how students are coerced from either receiving explicit images or feeling pressured into sharing explicit images to which puts students in an unsafe online environment in which they may feel uncomfortable or unable to cope with the situation at hand.

[15] http://www.lse.ac.uk/media@lse/documents/MPP/Sexting-Report-NSPCC.pdf

Current system[16]

The current system refers to the application of social media which is generally used today.

Facebook[17]



As of September 2018, Facebook has **2.23 billion** monthly active users.[18]

Features:

- You can create a profile to for personal or business use as it can be used to advertise a business
- You can create posts to show your friends
- You can stay in touch with peers and colleagues
- You can share videos and photos
- You can read the news or online articles
- You can direct message or group message individuals
- You can report/block individuals and remove friends from your friends list
- You can edit account settings to restrict what the public can see from your account

Faults of the system

- There is no profanity filter so you can't stop explicit content being sent to you

[16] https://www.linkedin.com/pulse/top-10-most-popular-social-networking-sites-apps-2017-rajiv-bajaj
[17] https://fitsmallbusiness.com/real-estate-facebook-page/
[18] https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

- There is no other means of counteracting cyberbullying besides removing the user as a friend, reporting the user or blocking the user
- The privacy setting to ensure that the content you post is not harmful to you in any way is very limited as although it restricts users from viewing your posts, it doesn't restrict friends from viewing your post and doesn't ensure that features like geotagging is off on images which could provide someone with your location. This could leave students susceptible to stalkers.
- There is a lack of safeguarding or awareness to adolescence on the network which could entail students being exposed to inappropriate content.

Twitter[19]

As of September 2018 there are **335 million** monthly active users of the service.[20]

Features:

- You can follow people to view their posts regularly or unfollow people
- You can post images or videos
- You can create a personalised or business profile
- You can repost what someone has posted before
- You can comment on someone's post

[19] https://slideplayer.com/slide/4495840/
[20] https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

- You can share the posts someone has made with other
- You can also block or report users
- There is a profanity filter which stops users from viewing content with any profane language

Faults of the system

- There is no way to avoid exposure to explicit content in the form or images or video
- There is no way to confront cyberbullying besides reporting an account which freezes the accounts inactivity
- The site is limiting when considering that you are only able to post content within 282 characters.

Instagram[21]

As of September 2018 there are 1 billion monthly active users with is up from 800 million in September of 2017. [22]

Features:

- Ability to use the service on smartphones and internet browsers
- You can watch someone's story which is a video or image that disappears after a day
- You can create stories of your own
- You can make a personalised or business profile

[21] https://www.curalate.com/blog/6-examples-and-best-practices-for-creating-instagram-ads/
[22] https://www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/

- You can also upload videos and images which are stored on the service
- You can like/comment/share posts
- You can follow individuals to view their content
- You can block/report individuals which freezes the accounts' activity until it is reviewed
- You can instant message individuals

Flaws of the system:

- There is no profanity filter or way to filter explicit content thus resulting in exposure of students to indecent content

YouTube[23]

As of September 2018 there are 1.8 billion users who use YouTube every month. YouTube is a video networking site which is made by Google. [24]

Features:

- Available for both smartphone and web browser
- Ability to watch movies, TV shows, cartoons, listen to music, watch private videos, vlogs and videos made by content creators on the site, watch live streaming video
- You can create a personalised account or channel to upload videos and earn money through monetisation on videos via revenue from advertisements
- You can follow channels to stay up to date with latest content releases
- You can contribute or take part in polls
- You can like/dislike/comment on/share videos
- You can save videos to watch later or download videos to view offline with a subscription service
- You can report content or comments for being explicit or harassing

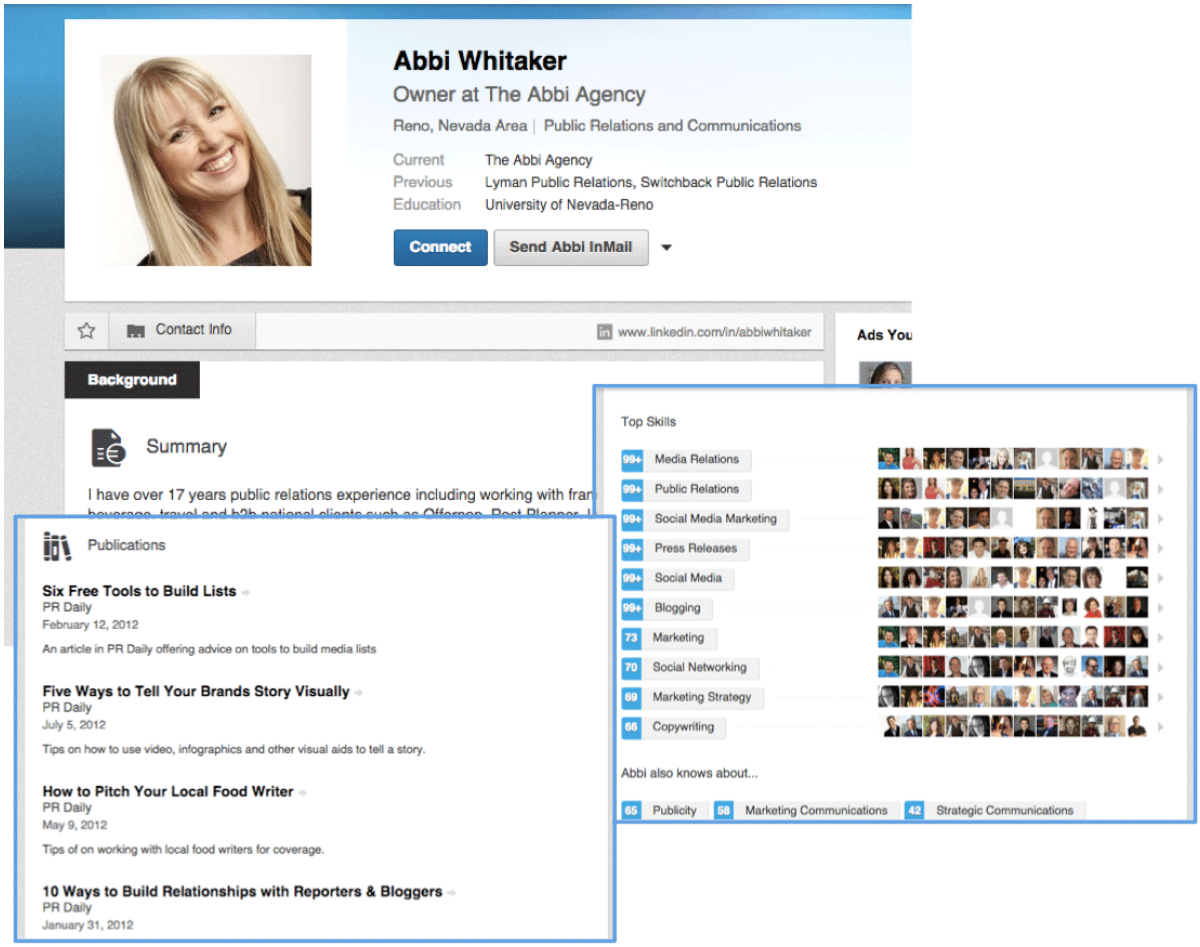
[23] https://www.koozai.com/blog/social-media/video-marketing/creating-optimising-a-new-youtube-one-channel-design/
[24] https://www.businessinsider.com/youtube-user-statistics-2018-5?r=UK

- You are also able to support a channel by donating funds
- There is a service aimed at children called YouTube kids that restricts explicit or profane content

Faults:

- There is no profanity filter or content filter on the original platform so if students use the original service they will be exposed to content that may be explicit or indecent
- Videos do not get removed from the service unless they have been reported by 3 different individuals or if a copyright holder has requested the service to stop the video from being able to produce revenue from advertisements as the content may be infringing copyright

LinkedIn[25]



As of September 2018 there are 500 million users of its business-focused network which is up from 467 million from October 2016. LinkedIn is a site for working professionals and business personnel.[26]

Features:

- You can make a public profile/resume by updating your profile with your qualifications, school attended, people you may know and any work experience you have completed
- Ability to help individuals find a job as well as helping businesses look for prospective employees by connecting professionals with mutual interests

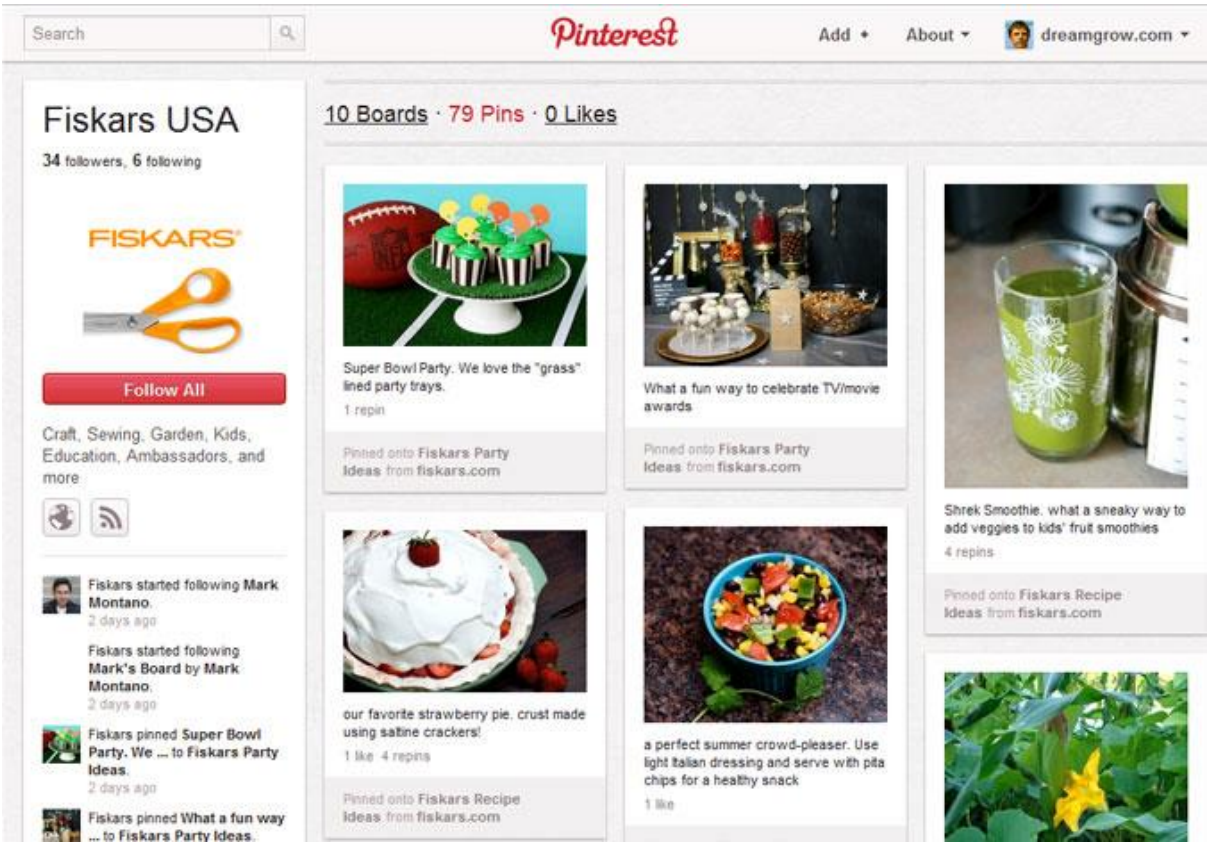
[25] https://komarketing.com/blog/10-examples-highly-impactful-linkedin-profiles/
[26] http://fortune.com/2017/04/24/linkedin-users/

- You can promote a business or advertise positions that are available.
- You can block/report individuals

Flaws:

- There is no profanity filter as the service is generally targeted at individuals seeing a job thus not being aimed at school students however, students may use the site to look for work experience thus putting them at risk to any indecent or profane language
- Although an individuals can be blocked/reported this doesn't stop the any indecent or explicit content that they have posted from being spread and can't be acted upon till the content is reported/blocked thus probability of exposure to this content is extremely likely

Pinterest[27]



As of September 2017, the platform had reached 200 million monthly active users which is up from 150 million in October 2016[28]. Pinterest is a similar social media platform to Instagram considering that it is a social media platform based on the sharing and posting of images. Its interface severely differs from Instagram and its main purpose is to act like a gallery or archive of images.

Features:

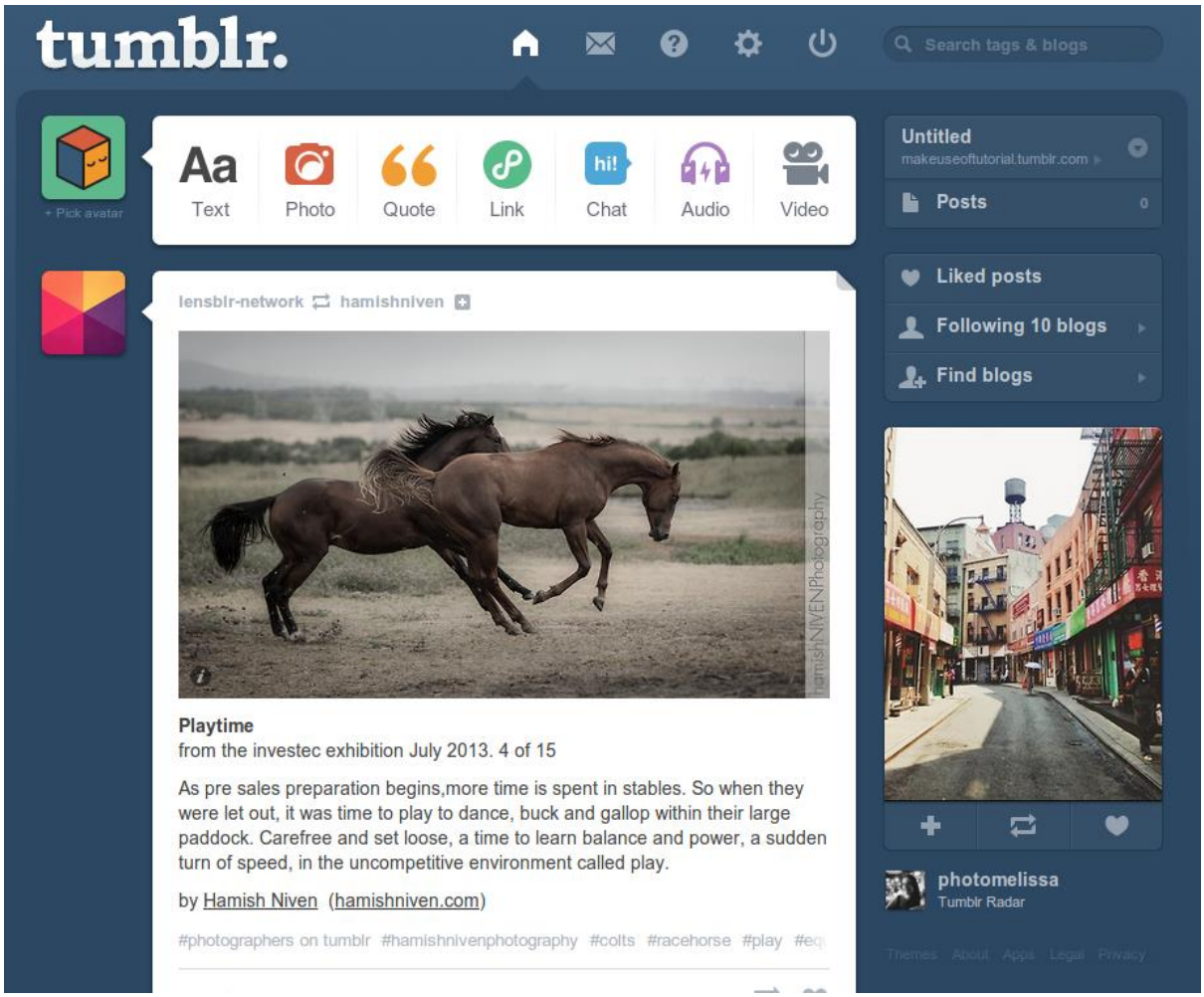
- You can save images to view later
- You can post/upload your own images
- You can share images with others
- You can block/report users for posting inappropriate content

[27] https://www.dreamgrow.com/41-great-examples-of-pinterest-brand-pages/
[28] https://www.statista.com/statistics/463353/pinterest-global-mau/

Flaws:

- There is no profanity filter so there is a risk of exposure to indecent or explicit content

Tumblr[29]



As of May 2013, there has been an estimated 30-50 million monthly users of the service which has dropped significantly from 300 million monthly users[30]. The network is used as a blogging platform.

Features:

- You can document your life in the form of a travel blog, photo blog, a blog to advertise your interest in a hobby or pastime or as a way to advertise a business
- You can message and video chat individuals privately by accessing the service from a web browser or smartphone
- You can like/dislike content on the service
- You can block users whether they are your friends or anonymous
- You can unfollow blogs
- You can delete/block or report individuals who follow your personal blog for continued harassment or spam

[29] https://www.makeuseof.com/tag/the-unofficial-beginners-guide-to-tumblr/
[30] https://www.businessinsider.com/tumblrs-active-users-lighter-than-expected-2013-5?IR=T

- There is an option to enable a profanity/content filter so that you aren't exposed to any explicit content while using the platform

Flaws:

- The profanity/content filter can be turned off very easily thus students may turn the feature off accidentally and be exposed to explicit or inappropriate content on the platform as there is no feature to ensure that when the content filter is turned off that it is done so by an individual who is of appropriate age.

Snapchat[31]

As of September 2018 there are 188 million daily active users of the service which is down from 191 million daily active users from January 2018[32].

Features:

- Ability to access service on a smartphone
- Ability for messages to disappear from recipients phone after a few seconds if the message is not saved
- Time stamps for when pictures which are taken in the application are sent to individuals on the service
- You can add friends from your devices contact list, from people who are nearby or by adding the users via their name or by their individual identification image
- Ability to overlay text or drawings on photos that you send or share with others
- Ability to voice/video call individuals and leave audio/video notes using the messenger feature
- Ability to use augmented reality filters which overlay on pictures you take

[31] https://neilpatel.com/blog/explore-snapchat-ads/
[32] https://www.statista.com/statistics/545967/snapchat-app-dau/#0

- You are able to post images/video which are viewable by anyone who is friends with you or who you specify which lasts 24 hours before being deleted
- Snapchat memories allow users to upload photos and videos from their devices photo library to the platform
- Ability to group message individuals
- You can find friends with an online map
- Ability to view posts created by the service or news outlets to keep users informed of current affairs
- Ability to block/report/unfriend users on the service and stop individuals from being able to view what you post or where you are located

Faults:

- If harassing content is received and not saved then evidence of the harassment is deleted by the service due to its messaging feature
- Feature to stop certain individuals from viewing your personal story or stopping individuals from locating you on the map provided by the service is very hard to locate which may result in lack of privacy of the user and put the user at risk to predators
- If you unfriend individuals they are still able to message you which may lead to users being exposed to harassment
- There is no profanity filter or way to monitor explicit content which may be received from individuals as a shared photo which may be harassing or abusive.

Data source and destinations

Name	Source	Destination
Signup details	New user of the system (student)	Account details database table
User information	Logged in user (student) filling in user information	User information database table
Post details	Logged in user (student) creating a new post	Post database table
Comment	Logged in user creating a new comment assigned to a post	Comment database table

Data volumes

Considering that there are approximately 1872 students from year 7 to 11 and approximately 200 students in post 16 (year 12 and 13), this brings the total students that could use my social network to 2072 with approximately 400 teachers that brings the maximum number of users who could use my network to approximately 2472 users. Due to the large quantity of users that may use the proposed system being more than 2000 people, to store all the users respective user information and their individual posts, comments and friends and any images that the use may upload must be very large to retain all of this data. Therefore, to store all of this data I would need a dedicated server to store all of this information with at least 8 to 16 GB of ram and at least a 1tb or 2tb hard drive to store all of this information.

However, as for this project I will only have approximately 6 users to test the system, a laptop such as my own or a server with at least 1GB of dedicated memory should be sufficient in storing any information that the users may store in the database or in the proposed solution.

Objectives

System objectives:

- Display homepage
- Redirect to admin login page or student login page
    - Validate admin login page
        - Redirect to admin page after successful login
    - Validation student login page
        - Redirect to profile page if logged in as student
- Redirect to signup page
    - Allow users to signup
        - Validate signup details by loading a file with banned words to see if the content in the form has inappropriate words
            - Store signup details in database
    - Redirect to login page after signing up
- Redirect to all student pages from profile page
    - Validate details entered on user information page
        - Store user information details in database
    - Validate created post by loading a file with banned words to see if the content in the post has inappropriate words.
        - store post details in database
- Redirect to all student pages from post page
    - Validate comment on a students' post by loading a file with banned words to see if the content in the comment has inappropriate words.
        - Store comment details in database
    - Display other posts that student can see
        - Delete post from database if students post is selected to be deleted
- Redirect to all student pages from friend page
    - Create relationship between users in database if they are friends
    - Remove relationship between users in database if they cease to be friends
- Validate pages accessed by students so that unauthorised users aren't able to access the social network
- Client server model to access social media network from anywhere with an internet connection.

User objectives:

- Must be able to use navigation to get to each page on the website
    - Index to admin login page
        - Users are able to see validation errors for admin login form
        - Admin login page to posts/comments/users
    - Index to student login page
        - Student login page to signup/profile/posts/friends/logout
- Allow users to sign up to social network
    - Students are able to see validation errors for sign up form
- Allow users to login to social network
    - Students are able to see validation errors for student login form
- student can view profiles

    - student to edit their own profile
    - students are able to view other users profiles
        - students aren't able to edit other users profile
- allow student to create posts
    - allow users to see validation errors on create post form
    - comment on posts
        - allow users to see validation errors on comment form
    - upload image associated with post
    - allow student to delete their own posts
        - ensure users are not able to delete posts of other users
    - allow student to search for posts
        - allow users to see form error if post that is searched does not exist
- allow student to add friends
    - allow student to search for friends to add
        - allow users to see form error if searched friend does not exist
    - allow student to delete friends
    - ensure students are not able to add or delete friends for other students
- admin permissions:
    - ability to delete student accounts
    - ability to delete student posts
    - ability to delete comments on students posts

Objectives added due to results from surveys

- Like and dislike functionality on posts
- Ability to block or report individuals as a student

Initial Data Dictionary

An initial data dictionary dictates how data is grouped and the purpose of the data being used in the interactive website by depicting how tables and data should work in the proposed solution.

Name	Purpose	Type	Size	Example Data	Validation
year	Store year group of student	Integer	2	11	Value must be an integer. Value can't be blank. Value must be between 7 and 11
interest	Store interest of student	string	12	Music	Value must be a string. Value must be any of the following: Music, Reading, Tv or Movies, Video Games, Art
postcode	Store postcode of student	string	7	SE256AE	Value must be a string. Value must be between 6 and 7 characters.
Town	Store town of student	String	40	Thornton Heath	Value must be a string
Date of birth	Store date of birth of student	Date/time	8	03/07/2001	Value must be date time. Value must be 8 characters

Title	Store title of post	String	200	Hi	Value must be a string. Can't include profanity in title.
Content	Store content of post	String	N/A	Welcome	Value must be a string. Can't include profanity in content
Image	Store image associated with post	Image	N/A	N/A	File must be an image or error is raised
Comment	store comment associated with post	String	N/A	Nice	Value must be a string. Can't include profanity in comment
Email	Store email of user	String	256	k.bhatti@hfed.net	Value must be a string. Value must be an hfed email. Value must be unique.
First name	Store first name of user	string	40	kamran	Value must be a string
Last name	Store last name of user	String	40	bhatti	Value must be a string
Username	Store username of user	string	40	Kam17	Value must be a string. Value must be unique
Password	Store password of user	String	40	Hg78234JHk	Value must be unique. Must be alphanumeric. Must be greater than 8 characters.

Name	Type	Start Value	Description
year	Integer	7	Keeps track of students year group
interest	string	Music	Keeps track of students interest
postcode	string	NULL	Keeps track of students postcode
Town	String	NULL	Keeps track of students town
Date	Date	01/09/2000	Keeps track of students date of birth
Title	String	NULL	Keeps track of title of students post
Content	String	NULL	Keeps track of content associated with students post
Image	Image	NULL	Keeps track of image associated with the students post
Comment	String	NULL	Keeps track of comments made by students associated with students posts
Email	String	NULL	Keeps track of email associated with a user

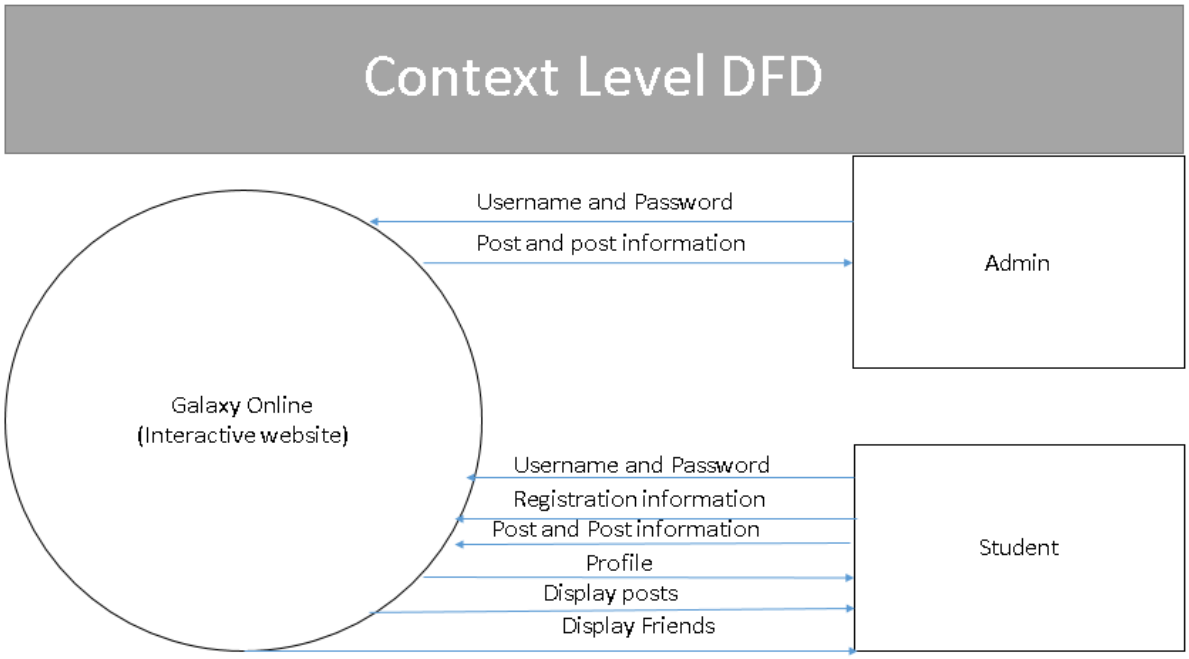
First name	string	NULL	Keeps track of first name associated with a user
Last name	String	NULL	Keeps track of last name associated with a user
Username	string	NULL	Keeps track of username associated with a user
Password	String	NULL	Keeps track of password associated with a user
Reported	Boolean	False	Keeps track of if the user is reported.

Initial Dataflow Diagram (DFD)

Dataflow diagrams portray how data flows through a system. In this instance I will be using data flow diagrams in means of depicting how data is used within an interactive website designed as a social media platform for the intention of being used within a school environment.

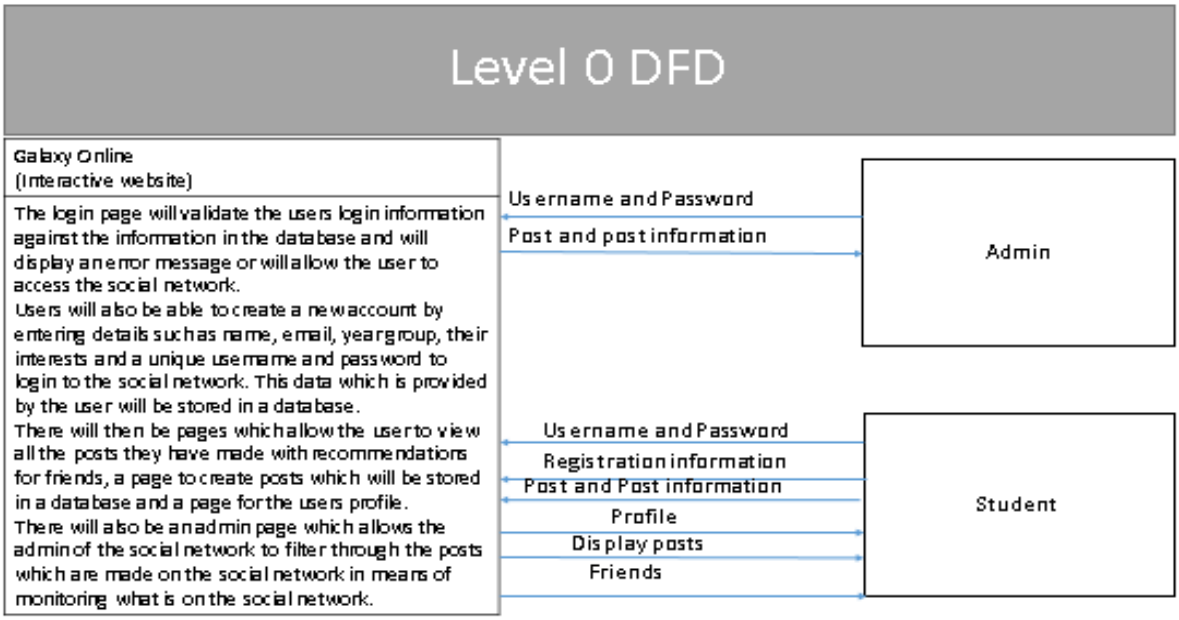
Context level DFD

A context level DFD is a simple diagram that gives an overview of the general processes of the interactive website.



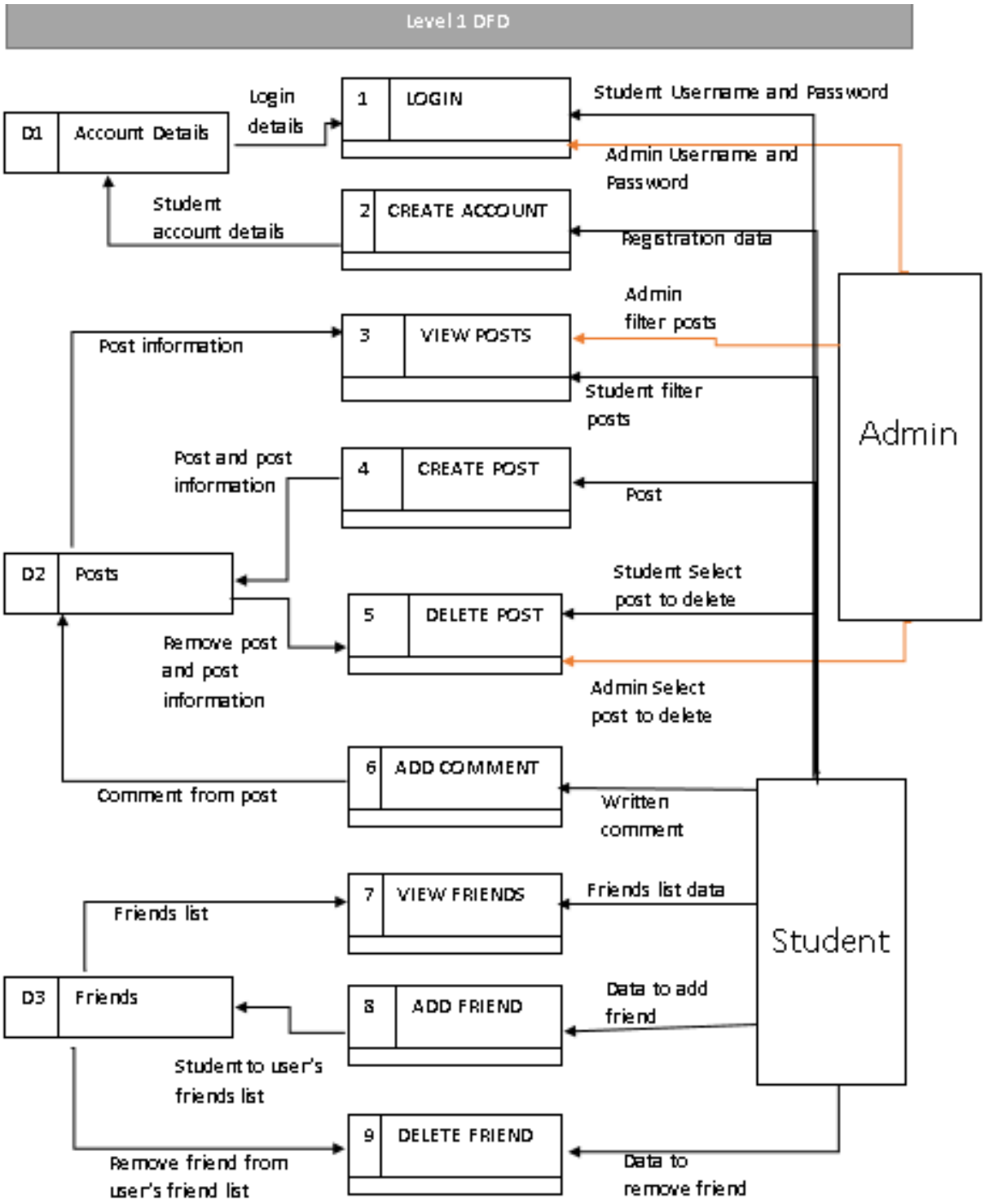
Level 0 DFD

A level 0 DFD is more detailed than the context level DFD as it shows how the data is used in the interactive website.



Level 1 DFD

A level 1 DFD is even further detailed than the context level DFD or level 0 DFD by displaying each individual process that the interactive website will carry out and how data flows from the users of the interactive website to the data stores being accessed by each process respectively.



Overview of processes

Process Number	Process Name	Description
1	LOGIN	When login details are entered these credentials are verified by checking the inputted details with the credentials already in the Account Details table. If the details match then a user is able to access the interactive website. If the details don't match then the user is denied access and asked to re-enter the correct credentials.
2	CREATE ACCOUNT	Registration information will be entered here such as username, password, name, year group, date of birth or email address to create an account for the user and is stored in the table Account Details.
3	VIEW POSTS	A user will select post to view which will then access the Posts table which stores multiple posts and will display the relevant information associated with the post.
4	CREATE POSTS	A user will create a post that will be stored in the Posts table that will store the posters' name and the relevant information like the posts file size, data type of the post, post description and when the post was created.
5	DELETE POSTS	A user or administrator will select a post to delete and then the post with its relevant information will be removed from the Posts table.
6	ADD COMMENT	A user will create a comment under a post therefore the publisher of comment, time comment was created and contents of comment would be stored in the Posts table.
7	VIEW FRIENDS	A student would select to view their personal friends list in which data would be pulled from the Friends table showing friends which the student has with relevant information like name and year group.
8	ADD FRIENDS	The student would select to add a friend thus the friends' information would be stored as a friend relationship in the Friends table.
9	DELETE FRIENDS	The student wants to remove a friend thus when selected the friends' information would be removed from having an association with the student from the Friends table.

Proposed Solution

Hardware requirements

System

The hardware requirements of the system would ideally be multiple servers which can host the solution. This would provide a smooth experience when on the network to allow users to experience a lag free interaction with the website. If multiple users were to use the proposed solution, there would be enough bandwidth on the server to support all the users.

Host	Positives	Negatives
Python Anywhere[33]	Allows for one web app. Allows for Python and MySQL consoles. Allows for private file storage.	Restricted out bound internet access from your applications with low CPU/bandwidth allocations. Low private file storage of 512MB. Limited options for free.
Amazon AWS[34]	Allows for multiple web applications. Provides multiple packages to use depending on preference. Provides NoSQL databases and storage packages. Ability to have 100GB of cloud storage for web application. Ability to combine packages	Package features can be confusing.
Open Shift[35]	Allows for one web app. 1 GB of memory allocation. 1GB of persistent storage allocation. Contains a blog which has lots of community support with tutorials, guides and information on how to deploy a web application.	Project sleeps after 30 minutes of inactivity and must sleep 18 hours in a 72 hour period thus, at peak intervals site may not be accessible if web application is put to sleep. Logs of server activity are not implemented as of yet.
Heroku[36]	Incorporates logs of server access. Allows for SSH access to server from a remote computer. Has guides to	Web application sleeps after 30 minutes of inactivity. Limited to 512MB of ram for the web application.

From my research it is shown that Open Shift is more popular as it is able to support many web frameworks and has support for many types of web application in addition to being able to support MySQL which I will be using to create the database however, I could use Heroku but I would have to migrate the MySQL database to Postgres. Although the Heroku service has inbuilt support for databases in the form of Postgres, Open Shift offers the functionality to use multiple types of databases such as MySQL, Postgres and SQLite. Additionally the server hosting available provides more storage space than Heroku which is invaluable for the web application I am choosing to create.

[33] https://www.pythonanywhere.com/pricing/
[34] https://aws.amazon.com/free/?nc2=h_ql_pr&awsf.Free%20Tier%20Types=categories%23featured
[35] https://www.openshift.com/products/pricing/
[36] https://www.heroku.com/pricing#postgres-pricing

Therefore I have chosen to use Open Shift to deploy the web application. If I have time at the end of creating the project I will deploy the web application on Open Shift.

End User requirements:

The hardware requirements of the user would ideally be any digital device with an internet connection which would be optimised for each device to allow a smooth interaction with the proposed solution. This would entail programming the solution to utilise the hardware specifications of the device it is being utilised on to the fullest.

Software requirements

There are many programming languages that I can utilise to produce the proposed solution.

Language to make the web application	Features	Faults
PHP[37]	Ability to use PHP to connect to database. Designed specifically for web applications. Lots of support for development of web applications from community who use PHP.	Library support is very limited in comparison to Flask and Django.
Flask[38]	Ability to customise URL's of webpages. Tools to help process requests. You can make SQL queries through SQLAlchemy. You can structure the project however you like allowing you to fully customise the project.	Project is not structured so if something is needed to be changed you may have to change how the whole web application is structured. Documentation on how to customise URL's is very confusing.
Django[39]	Project is structured and in a layout that is easy to follow and amend. Ability to customise URL's of webpages. Tools to help process requests. Ability to use Django's custom SQL queries. You can use SQLAlchemy instead of the inbuilt SQL queries that Django offers.	There is a lack of customisation with how your coding is structured. If the SQL queries you need to perform don't match Django's custom queries then you have to code the SQL queries with the raw function. If you use SQLAlchemy then the admin page provided by Django ceases to function correctly.

I can conclude that the best language for me to use is Django or Flask as PHP is difficult to use considering my coding ability and due to me being accustomed to using python which is the programming language used by Flask or Django. I decided to use Django as the documentation for the language is thorough and easy to follow in contrast to Flask's documentation which is quite cumbersome in regards to content. Additionally, Django has many examples which I can reference from the documentation which I can modify for use in my project in contrast to Flask which has some examples but doesn't fully explain how they work. Furthermore, the structure of a Django web application will help ensure that if I need to, I will be able to edit or improve my web application whilst still having a structure to my code in contrast to Flask where if I were to edit or improve my

[37] https://hackr.io/blog/python-vs-php-in-2019
[38] http://flask.pocoo.org/docs/1.0/patterns/sqlalchemy/
[39] https://gearheart.io/blog/flask-vs-django-which-is-better-for-your-web-app/

code I could possibly have to change multiple attributes of the web application due to its lack of structure.

Additionally I will have to decide on a database to use[40]:

Database name	Positives	Negatives
SQLite	The entire database consists of a single file thus making the database extremely portable	Not suitable for multiple users who may access the database. Limited throughput as only allows a single write operating to take place at a given time.
Postgres	Wide community support. Very reliable in comparison to MySQL. Easy to migrate database systems.	Can be less performant in comparison to MySQL for read-heavy operations. May be difficult to get support as it isn't very popular. Difficult to find hosts or service providers that support the database. Can be very time consuming in comparison to MySQL.
MySQL	Can be installed very easily. Lots of SQL functionality. Has security features built in. Easy to use and manage.	Does not implement the full SQL standard. Can lack full-text search depending on database engine

Django uses SQLite by default. Although SQLite is a single file and can be extremely portable it is not suitable for multiple users who may access the database and is very limited in the type of SQL queries you can. Additionally, MySQL can be installed very easily and has the ability to use many SQL functions but, it lacks some basic SQL functions and can also be very difficult to manipulate data correctly as it doesn't handle NULL functions correctly without a great deal of configuration. Therefore, to save time on configuring MySQL to work correctly and while still retaining the ability to use SQL queries and functions, I have chosen to use SQLite as it is more reliable that MySQL. Using Postgres can be very time consuming and the documentation on how to use SQL functions with Postgres is very easy to follow therefore I can reference where I have modified queries from the documentation when creating my web application. However, SQLite is straightforward and easy to use thus I will be using it for my database as I will most likely not need to use queries beyond my basic knowledge of SQL databases. Thus I will be using SQLite.

Project Limitations

Hardware limitations

Due to the funds I have at hand, I am unable to use multiple servers to host the social network. Therefore, I will have to host the solution on a third party servers or on my computer. This may cause slowdowns at peak intervals when the network is subject to a multitude of users as the host of the third party servers may have restrictions for free use. Additionally, there may be a limitation on the amount of people who may be able to access the social network at a given time and there may be a threshold to the number of people who are able to use the social network due to the storage

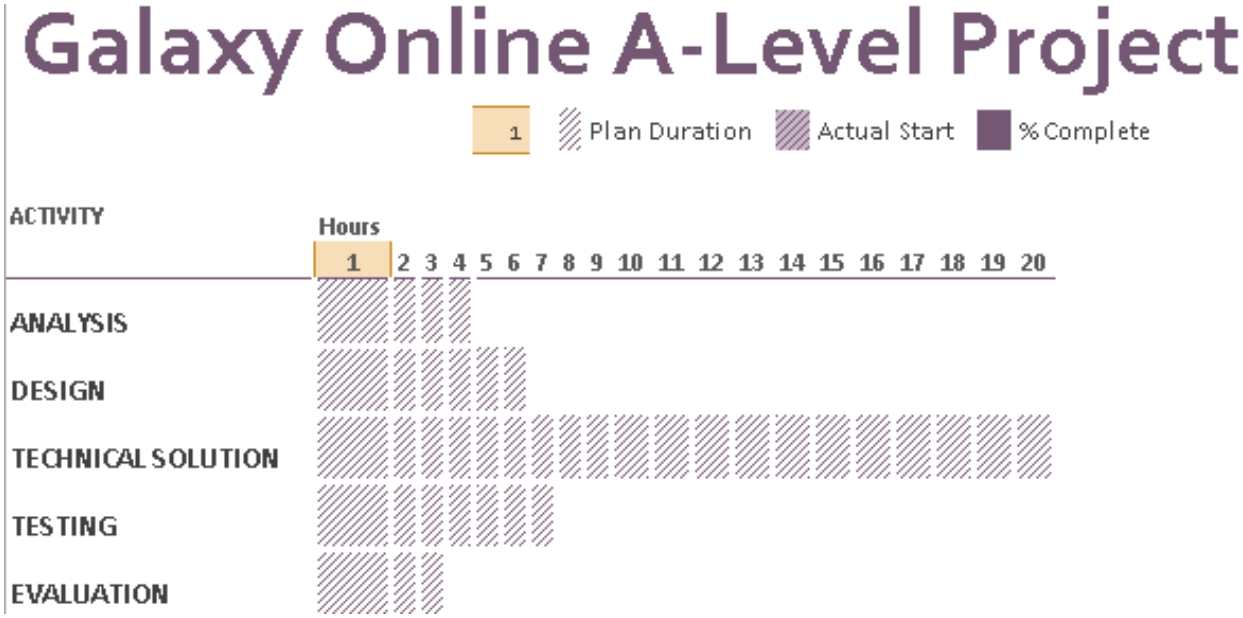
[40] https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems

size available from the hosting service for the web application. However, for development using a server hosted on my computer is acceptable at this time.

Software limitations

Due to my programming ability and time constraints I will only able to produce a solution which is functional in web based format and not an application. I will also be unable to produce a project which is functioning on a mobile platform as I lack the skills to develop a project on a mobile platform and I lack the skills to utilise the hardware of a mobile platform to its potential to allow a smooth experience on the platform. Due to the way in which Django is programmed, using SQL functions may be difficult however, coding the SQL functions in their raw format will work as normal. Additionally, due to the way in which SQLite is designed, database access or queries from a database may be extremely simplified in comparison to MySQL or PostgreSQL.

Gantt chart



DESIGN

Overview

This section will depict how my proposed solution will look and the steps I will take to design the proposed system.

Storage media

As users can upload a multitude of posts to their account and are able to sign up to the system, approximately 2500 people could use the proposed solution. If a user posts approximately 5 posts per day on the project and each image on the post takes an average file size of 100 kb; the approximate storage space taken up by posts is 250,000 KB or 245 MB. However, as I am using the project in the developmental stage, an allocated size of 1GB is enough.

There are 2 ways I can distribute the software: as an executable file or a web application. If I were to choose to distribute the file as an executable on a CD-ROM, USB drive or DVD rom, to distribute the executable to end users would be extremely expensive n in a school environment and would not account for users who may lose the media drive in which the executable is stored on. Although this would allow users to store a greater amount of data on the application, allowing other users to view the posts made on the flash media would be very difficult to which I don't have the programming capabilities to carry out. Additionally, if the drive were taken by someone, they could view the images that you have posted on the network which could infringe on your privacy.

Thus, I propose to use a web application for distributing my software. This will allow users to upload posts a network and will increase security of the network as no data that the user submits on the network will be stored on the client's computer only on the host server. Furthermore, the project would only be limited by the internet connection of the system to the host server and not limited by the hardware requirements of the client's computer which would be the case for an executable application on a storage media device.

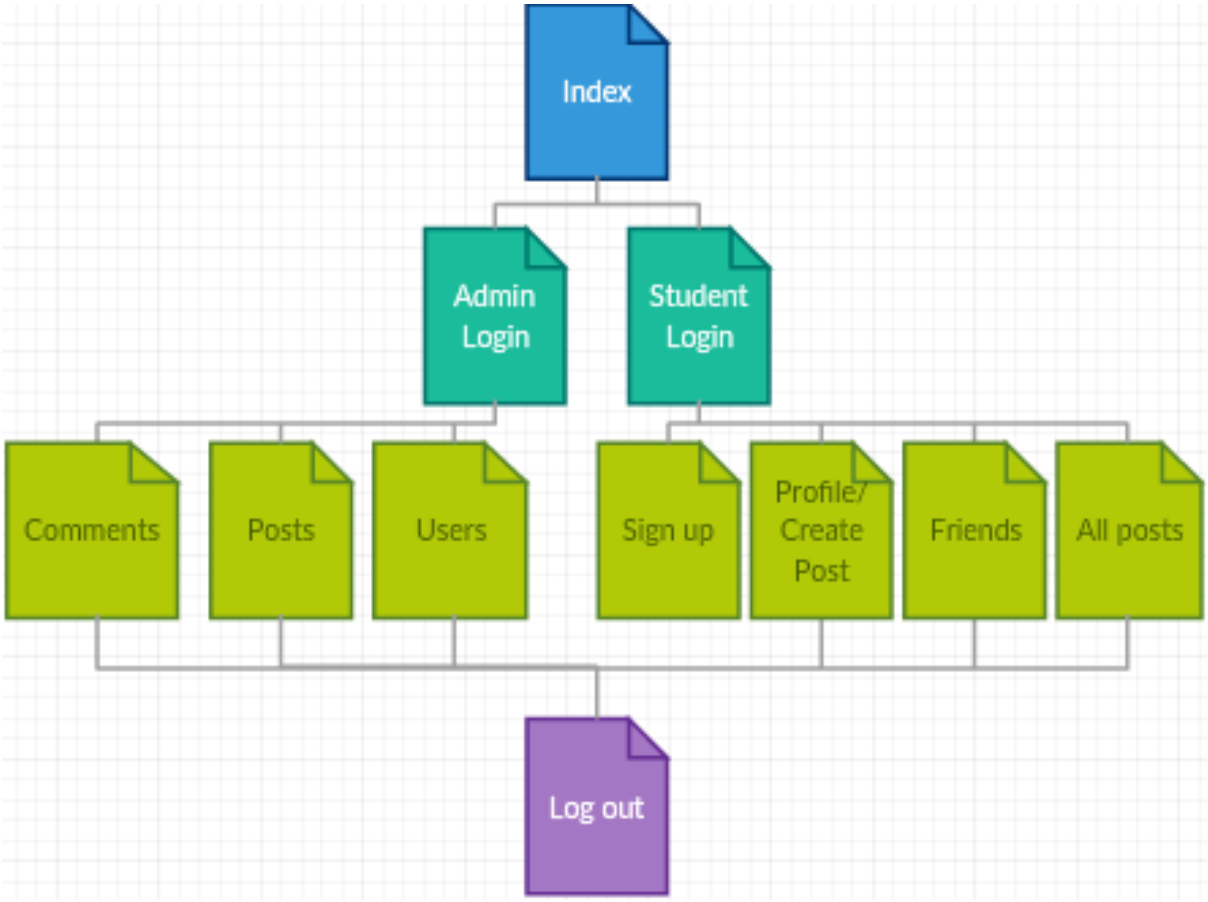
Input	Process	Storage	Output
Signup details	Check if details are in database. If not in database and valid proceed else raise error	Database table: Account details	Redirect to login page or display error if form is not valid
Login details	Check if details are in database. if in database and valid proceed else raise error	Database table: Account details	Redirect to profile page or display error if form is not valid
User information	Check if form is valid else raise error	Database tables: User Information Year Group Interest Postcode Town	Redirect back to profile page or display error if form is not valid
Post details	Check if form is valid else raise error	Database table: Posts Date And Time	Redirect back to post page or display error if form is not valid

Comment details	Check if form is valid else raise error	Database table: Comments Date And Time	Redirect back to the post or display error if form is not valid

This details any physical inputs that the user will enter into the proposed system to be stored in the database and the outputs that will be displayed on screen.

Webpage Navigation Chart[41]

This section will depict how the user will navigate through the website.

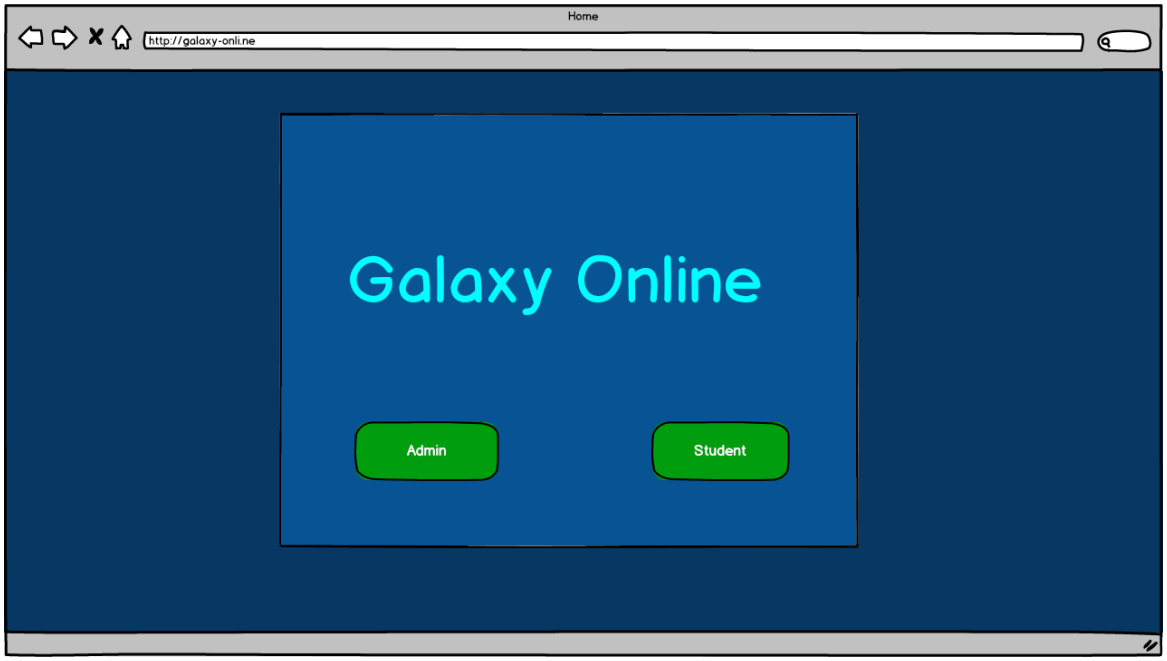


Although the user is able to navigate the website in this hierarchical structure, I plan to allow a student to access the profile/create post page, friends page and all posts pages from each other through buttons. I have formatted the layout of the webpage navigation in the format to avoid causing confusion in how the web application is structured.

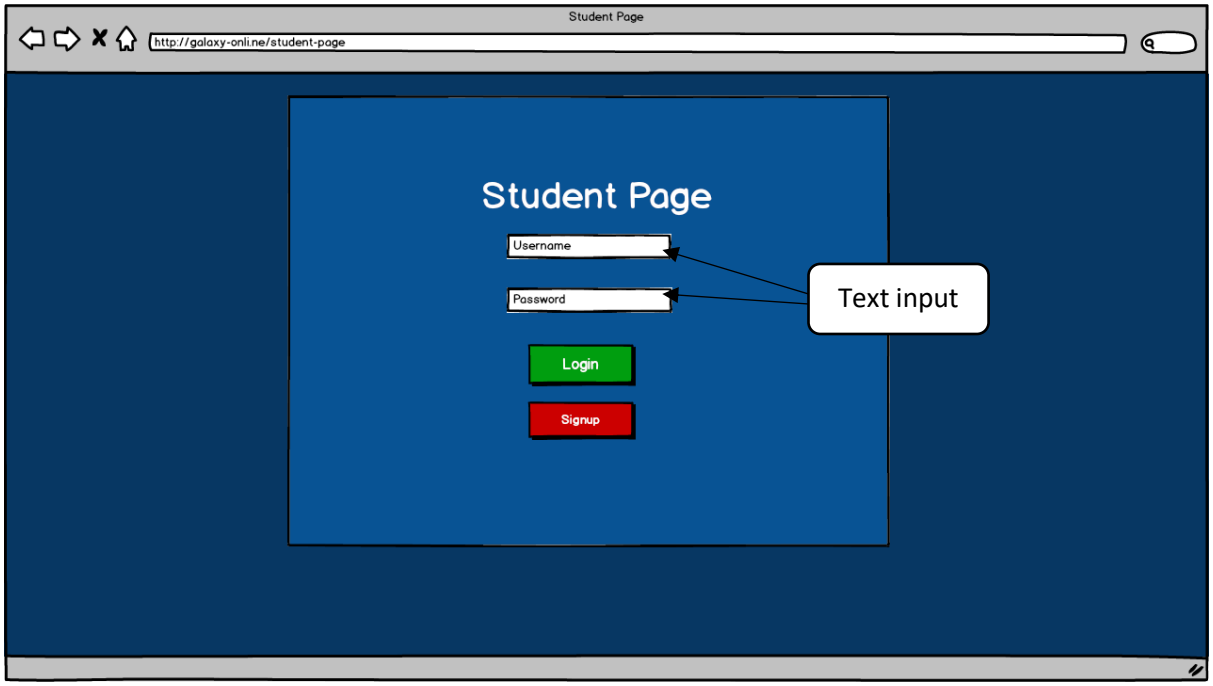
[41] https://creately.com/diagram/example/go2ewdki1/Marketing+Web+Sitemap+Template

Graphical User Interface[42]

These are wireframes which I have designed in order to create an idea of how I would like to end product of my proposed solution to look.

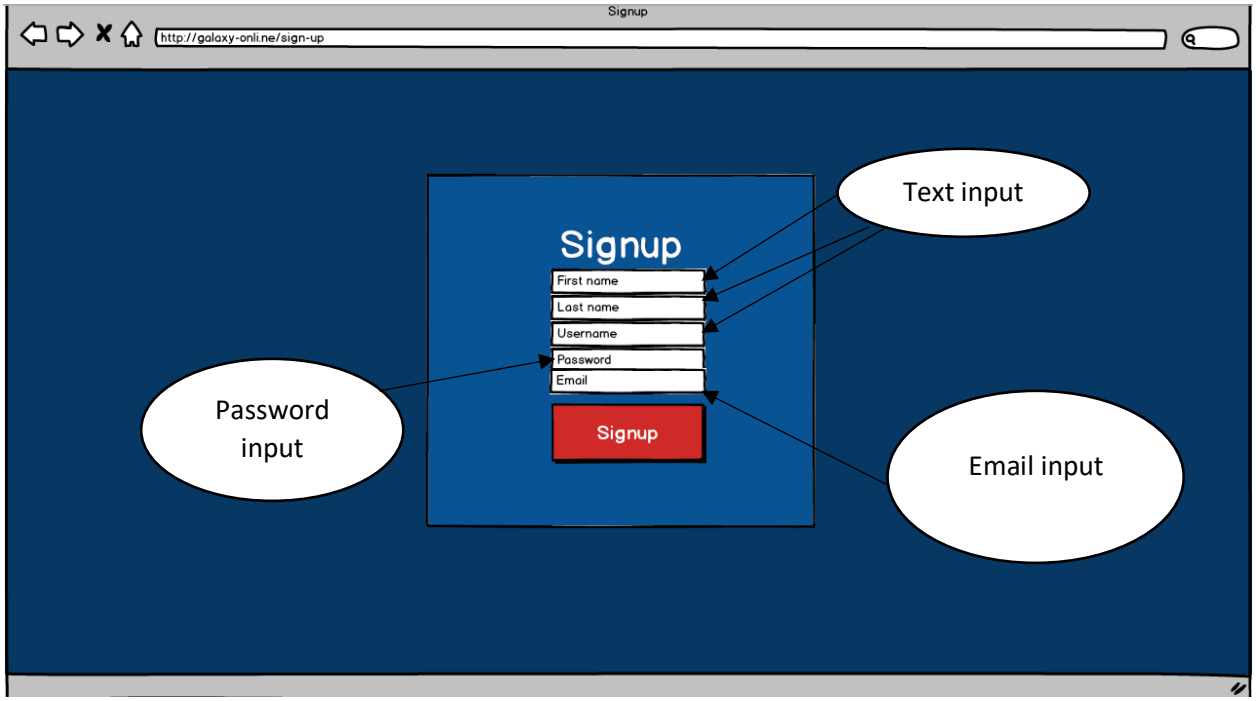


I designed the page like this so that the buttons on the page are easy to view and directly in the line of site of the user of the website. The colour scheme was decided by me so that the website looks calming to use as the colour blue has calming effects on those who look at it. I have created a box around the buttons and title of the page being Galaxy Online so that the main context on the page seems highlighted. I decided on using green buttons for the buttons on the page so that there is a contrast with the main scheme of the website so the buttons stand out on the page.

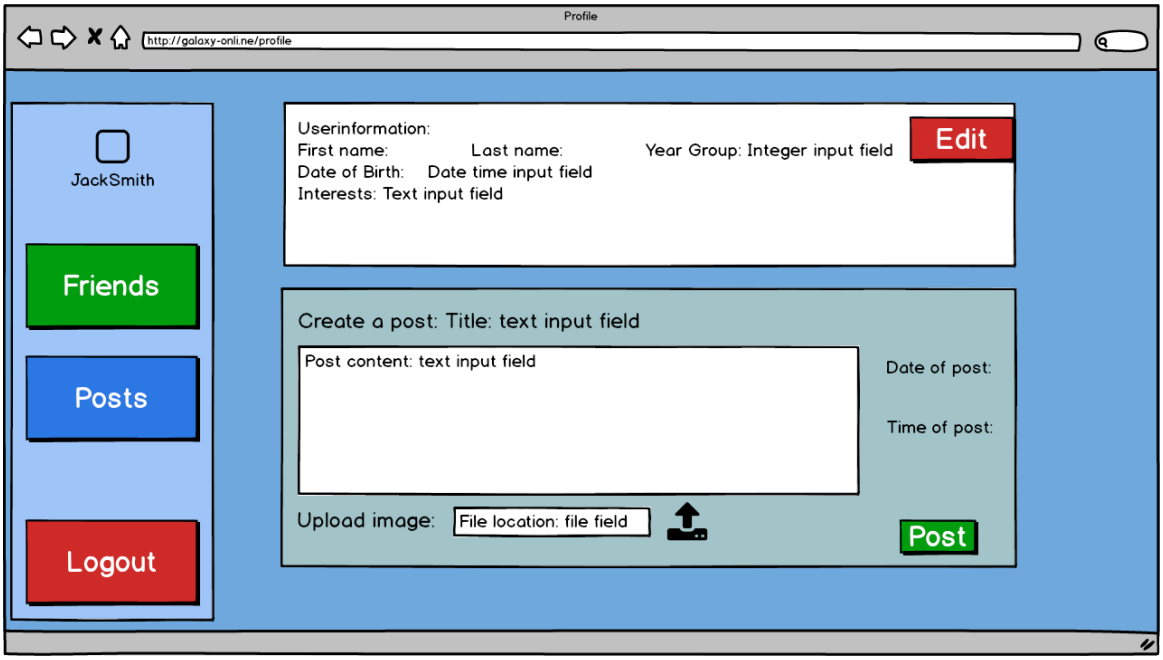


[42] https://balsamiq.com/wireframes/

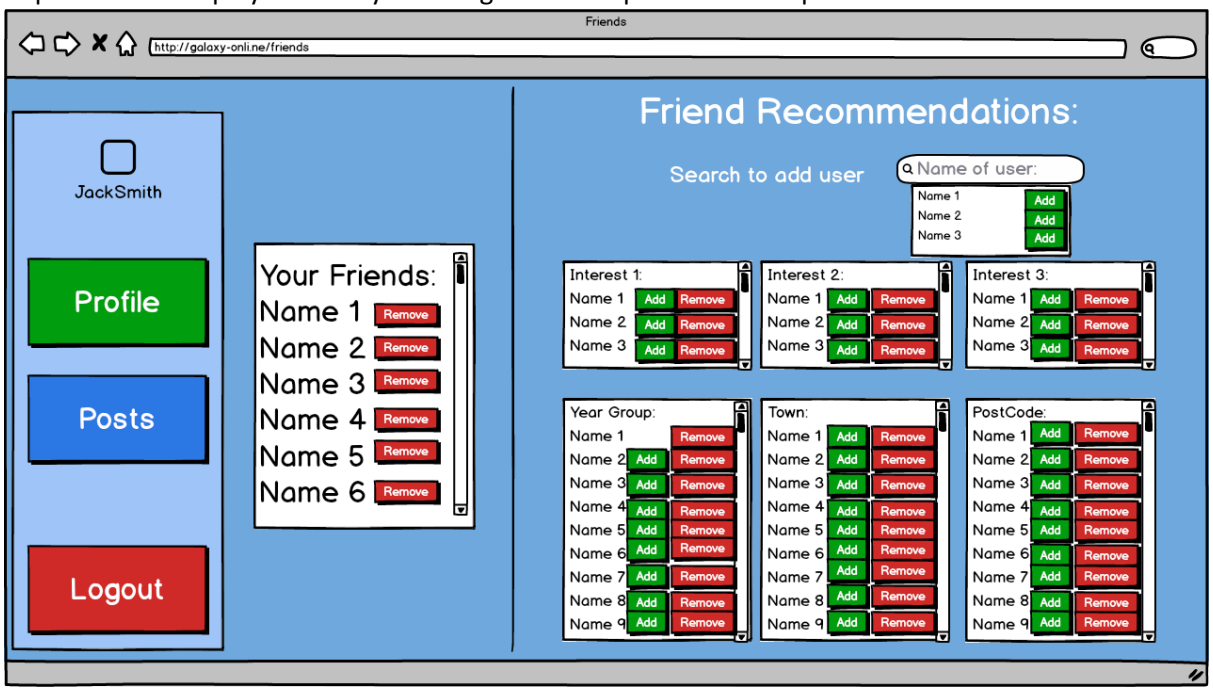
The student page has the same colour scheme as the main page. The title of the page is in white so that there is a contrast with the container around the content of the page. The buttons on the page are green and red to allow users to be able to differentiate between the login and signup buttons easily. The text and password inputs on the page are in the centre of the container for the main content on the page and are text inputs for the username and password fields on the page.



This page also has the same colour scheme of the student and homepage. The main container is a different colour to the background to show contrast so that the main content is clearly visible. The title of the page is white so that the user can clearly see what page they are accessing. The fields on the page for first name, last name, username, password, email. First name, last name and username will be text input fields. Email will be an email field and password will be a password field.

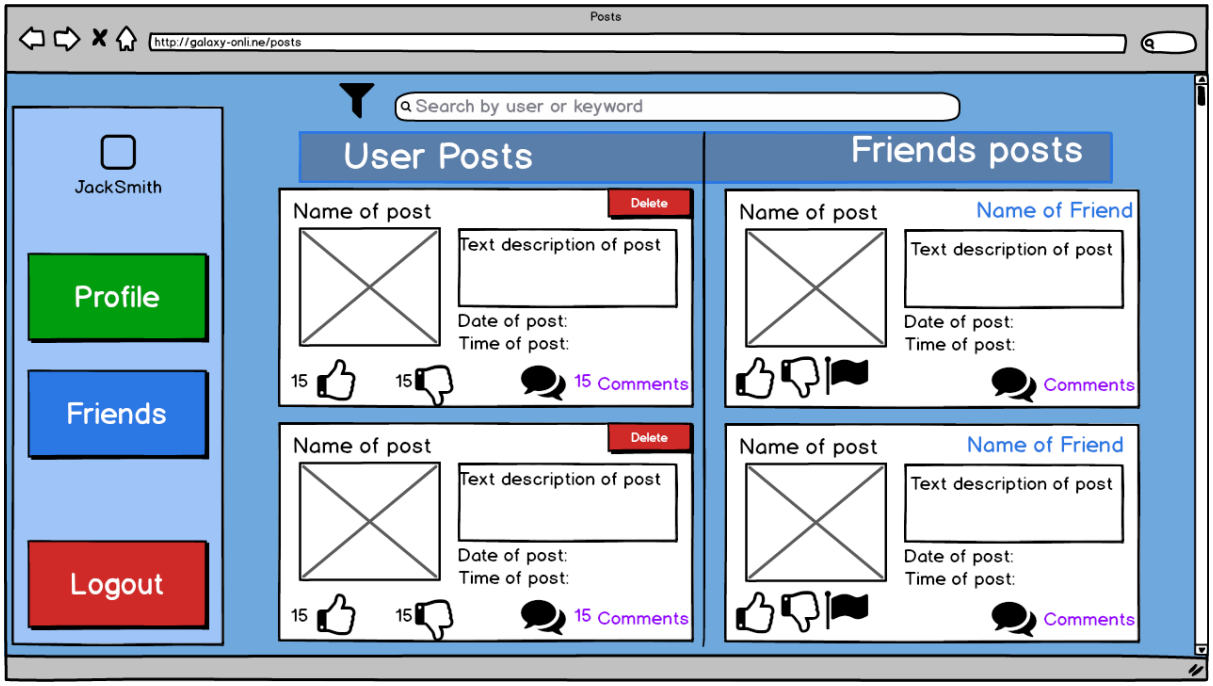


This is the profile page which the user will be directed to after signing into the social network. This page has a light blue background to show a contrast with the login pages form before. The top left corner will show the name of the user logged into the page. The buttons are different colours so that you can differentiate between the buttons on the page easily. The logout button is red so that the user is able to tell that the button is completely different to the other buttons on the page and because the colour red connotes warning so users will inherently know that the button will lead to logout as many pages have the logout button in red. This page displays the user's information with their first name, last name, and year group, date of birth and interests which can be edited from this screen. Clicking edit will display text input fields to edit interests, town and postcode whilst date/time input fields will be used for date of birth and an integer field for year group. The create a post section allows users to create a post with time stamped information on the post with the relevant posts content and allows users to upload an image along with the post. The post button is green so that the user knows that when clicked the button will post the content in the relevant fields. The user information and create a post field are white as to be able to distinguish with the background of the page. The title of the post and post contents will be a text input field. The file location field will just be an input to get the file location on the clients' computer. The date and time of post will be displayed as they are assigned to the post when the post is created.



This is the Friends page which the user will be directed to after clicking the friends' button. This page has a light blue background to show a contrast with the login pages form before. The top left corner will show the user who is logged into the page. The buttons on the side are persistent so the buttons can be accessed from wherever you are on the page. The buttons are different colours so that you can differentiate between the buttons on the page easily. The logout button is red so that the user is able to tell that the button is completely different to the other buttons on the page and because the colour red connotes warning so users will inherently know that the button will lead to logout as many pages have the logout button in red. From this page you are able to see your friends which you can remove with the red button next to the name of the friend. You will also be able to search for users to add by their name on the social network and you can add friends which are recommended y similar interest as the user, same year group, town and postcode. The search field is a text input field. To add users the button is green and to remove users the button is red. To view more users

than the users that are displayed on the page you can use the scroll feature next to each field. The background of each of these lists are white as to clearly see the list in contrast to the original background of the page.



This is the Posts page which the user will be directed to after clicking the posts button. This page has a light blue background to show a contrast with the login pages form before. The top left corner will show the user who is logged into the page. The buttons on the side are persistent so the buttons can be accessed from wherever you are on the page. The buttons are different colours so that you can differentiate between the buttons on the page easily. The logout button is red so that the user is able to tell that the button is completely different to the other buttons on the page and because the colour red connotes warning so users will inherently know that the button will lead to logout as many pages have the logout button in red. The search function is a text input which has a filter next to the field to filter the posts that are shown on the page. The user posts and friends' posts headings and the search field will be persistent on the page so that when scrolling through posts, the user will be able to show which posts where made by the user and which posts are made by the users' friends. The posts on the page will display the name of the post, number of likes, dislikes, and comments with a time stamp for each post with their relevant description with the image of the post if the user posted an image with their relevant post. For user posts there will be a red button in the top corner of the post to allow users to delete the post that they created. For posts created by friends of the user the post will have the name of the friend who made the post in the top right corner and will have relevant buttons such as: like, dislike, comment and flag for flagging posts for inappropriate content however, these buttons which have like, dislike and comment don't have the number of users who have clicked these relevant buttons. These are only able to be viewed by the users who created the post not the friends who viewed the post.

Data Dictionary

Name	Validation checks	Description	Error message	Data	Caught
Email	Presence	Make sure email they have a sensible hfed email	This web app only allows hfed users	kamran@gmail.com	Yes
		Make sure email is unique to all users	That email is already assigned to a user	k.bhatti@hfed.net	Yes given that a user already has that email
First Name	Presence	Make sure name is a suitable length, present and has no numbers	Enter name/ remove number from name	Kamran7	Yes given that I implement a feature to detect is a number is in the first name and last name
Last Name				Bhatti8	
Username	Presence	Make sure username is unique, of suitable length and presence	Enter username/ Username is already used	Kamran7	Yes given that the username has already been taken
Password	8<=length<=16	Make sure password is not common and of valid length	Enter password/ Password is too common	abc123	Yes as Django will handle any passwords that are common and not of correct length
Year	List (“7”,”8”,”9”,”10”,”11”, ”12”,”13”)	Only allow years in list	Enter year group	14	Yes as not in list
Interest	List("Music", ”Reading, "Tv or Movies”, "Video Games”, "Art"	Only allow interests in list	Enter interest	Book	Yes as not in list
Postcode	6<=length<=7	Make sure postcode is of valid length	Enter postcode	BDE	Yes as not of correct length

Town	Presence	Make sure town is present	Enter town	A	No as can't catch this error as we don't know whether a valid town is inputted only that the field has been filled
Post title	Presence	Make sure value is filled and not contains profanity	Enter in field/ remove profanity from field	<profane language>	Yes
Post content	Presence				
Comment on post	Presence				
Post image	Datatype(image)	Make sure file uploaded is an image	File is not an image or may be corrupt	<file name>.docx	Yes

Relational Database Design

This will depict how the database will be modelled.

Un-normalised form

This is how the tables in the database will look un-normalised. This is when the data in the database is not identified by a primary key.

UserInformation (Name, DateOfBirth, YearGroup, Interests, Address, LoginInformation, Posts, Comments, Relationships)

This stores the data name, date of birth, year group, interests, address, username, password, posts, comments and relationships

UserInformation
Attributes: Name DateOfBirth YearGroup Interests Address LoginInformation Posts Comments Relationships

First normal form

This is when the data is structured in a way that no data is repeated and only one value per field thus making the data atomic and that each record is unique. This means I will start to use primary keys to organise how the data is structured in the database.

AccountDetails (AccountID, FirstName, LastName, DateOfBirth, YearGroup, Interests, PostCode, Town, Username, Password, Email, ContentPosted, DateOfPost, TimeOfPost, ContentCommented, DateOfComment, TimeOfComment, StartOfRelationship, Reported)

This stores first name, last name, date of birth, year group, interests, postcode, town, username, password, email, content posted, date of post, time of post, content commented, date of comment, time of comment and start of relationship. This will also store if the user is reported.

AccountDetails
Attributes:
AccountID
FirstName
LastName
DateOfBirth
YearGroup
Interests
PostCode
Town
Username
Password
Email
ContentPosted
DateOfPost
TimeOfPost
ContentCommented
DateOfComment
TimeOfComment
StartOfRelationship
Reported

Second normal form

This depicts the data stores in a way that meets the requirements of first normal form but all non-key attributes should depend on all parts of the primary key which can be achieved by creating more tables.

AccountDetails (AccountID, FirstName, LastName, DateOfBirth, YearGroup, Interests, PostCode, Town, Username, Password, Email, Reported)

This is the entity **AccountDetails**. This will be a data store in the database which will store the name of the user, date of birth, year group, interests, address, preferred username, password and if the user is reported. For this entity AccountID will act as the primary key to be a unique identifier to the users of the social network.

Posts (PostID, AccountID (**foreign key**), ContentPosted, DateOfPost, TimeOfPost, NumberOfLikes, NumberOfDislikes, ContentCommented, DateOfComment, TimeOfComment)

This is the entity **Posts** which will act as a table in the database to store content that is posted, the date of the post and time of the post and comments created for the post with the date and time of when the post was commented with number of likes and dislikes on posts. The PostID acts as the primary key to identify each individual post created by a user. The AccountID acts as the foreign key of the table as to assign each individual user to the posts that they create

Friends (FriendsID, AccountID (**foreign key**), StartOfRelationships)

This is the entity **Friends** which will act as a table in the database to store the friends that the user has under Relationships. FriendsID acts as the primary key to identify the friend of the user where AccountID acts as the foreign key to assign each relationship to the specified accounts for each friend.

Friends
Attributes: FriendsID AccountID (**foreign key)** StartOfRelationship

AccountDetails
Attributes: AccountID FirstName LastName DateOfBirth YearGroup Interests PostCode Town Username Password Email Reported

Posts
Attributes: PostID AccountID (**foreign key)** ContentPosted DateOfPost TimeOfPost NumberOfLikes NumberOfDislikes ContentCommented DateOfComment TimeOfComment

Third normal form

This meets the same requirements as second normal form and ensures that non-key attributes should not depend on other non-key attributes.

UserInformation (UserID, Firstname, Lastname, AccountID (**foreign key),** DateOfBirth, YearID (**foreign key**), InterestsID (**foreign key)**, PostCodeID (**foreign key**), TownID (**foreign key)**)

This is the **UserInformation** table which will be used to generate a profile page for the user on the website. The primary key of this table is UserID which will store DateOfBirth, Firstname and Lastname. The foreign keys for the table are AccountID, YearID, InterestsID, PostCodeID, and TownID. These foreign keys are used to gather the corresponding data from other the other tables to be used within this table. I have done it this way as AccountID doesn't need to depend on the UserInformation.

AccountDetails (AccountID, Username, Password, Email, Reported)

This is the **AccountDetails** table which will store the users' username and password of which they specify and will also store the users and email and if the user is reported. This uses the primary key AccountID.

YearGroup (YearID, Year)

This is the **YearGroup** table which will store the individual years of the students with the primary key YearID to act as the identifier of each record in the table. This is a separate table as the only year groups applicable are 7, 8, 9, 10, 11, 12 and 13 and as the data would be repeated in each record, having a table would allow me to minimise the amount of repeated data in the **UserInformation**

table and so that I could use SQL commands to validate that the year group of the user specified is within the table without having to use general IF or ELIF statements using the python implementation in Django.

Interests (InterestID, Interest1, Interest2, Interest3)

This is the **Interests** table which will be used to store the name of each interest and will be identified by the primary key InterestID. This is to ensure that the interest that the student inputs is already within the specified list and that when recommending friends to users in the technical solution, I am able to recommend individuals without data having to be repeated in the **UserInformation** table.

PostCode (PostCodeID, Postcode)

This is the **PostCode** table which will store the postcode of each user and will be identified by the primary key PostCodeID. This is a separate table as many students will have the same postcode, I have decided to create this table to limit the amount of data that is repeated in this field when in the **UserInformation** table. This improves the efficiency of the database as instead of iterating through the database to recommend friends based on the postcode entered, I can use the primary key to identify the postcode which decreases the time to process the statement to filter the table for a specific postcode.

Town (TownID, TownName)

This is the **Town** table which will contain the name of the town and will be identified by the primary key TownID. This is a separate table as many students may live in the same town so to limit the amount of data that is repeated, the primary key would be used instead of the town name. This will make recommending friends based on town easier as I would be able to identify the town in the database via the primary key instead of iterating through the table to see if the inputted variable for town matches thus improving the efficiency of the database.

Comments (CommentsID, PostsID (**foreign key**), UserID (**foreign key**), ContentCommented, TimeID (**foreign key**))

This is the **Comments** table which will contain the content commented and the date of comment and time of comment from the foreign key TimeID. The unique identifier for each comment will be the primary key CommentsID which will use the foreign key UserID to assign the comment to the user who created the comment and will use the foreign key PostID to assign the comment to the post that the comment is referring to.

Posts (PostsID, UserID (**foreign key**), Title, ContentPosted, TimeID (**foreign key**), LikesID (**foreign key**), DislikesID (**foreign key**))

This is the **Posts** table which will be used to store the data of the title, content posted, date of comment and time of comment from the foreign key TimeID. The foreign key UserID would be used to assign the post to the user who created the post whereas the unique identifier PostsID which is the primary key of the table will be used to distinguish between posts in the table. The foreign keys LikesID and DislikesID with link to the relative like and dislike values of each post.

Friends (FriendsID, UserID (**foreign key**)

This is the **Friends** table which will use the primary key FriendsID to act as the unique identifier for each record with the foreign keys UserID, to identify individuals and their friends.

Likes (LikeID, NumberOfLikes)

This is the **Likes** table which will use the primary key LikesID to act as a unique identifier for each record of each value of Like. The number of like's field will store the integer value of the number of likes

Dislikes (DislikeID, NumberOfDislikes)

This is the **Dislikes** table which will use the primary key DislikeID to act as a unique identifier for each of the records for each value of the integer value stored in number of dislikes.

DateAndTime (TimeID, Created)

This is the **DateAndTime** table which will use the primary key TimeID as a unique identifier for each value of Created that a comment or post will be made.

UserInformation
Attributes: UserID FirstName LastName AccountID (**foreign key)** DateOfBirth YearID (**foreign key**) InterestsID (**foreign key)** PostCodeID (**foreign key**) TownID (**foreign key)**

AccountDetails
Attributes: AccountID Username Password Email Reported

Friends
Attributes: FriendsID UserID (**foreign key**)

Interests
Attributes: InterestID Interest

PostCode
Attributes: PostCodeID Postcode

Town
Attributes: TownID TownName

Comments
Attributes: CommentsID PostsID (**foreign key**) UserID (**foreign key**) ContentCommented TimeID(**foreign key**)

Posts
Attributes: PostsID UserID (**foreign key**) Title ContentPosted TimeID(**foreign key**) LikesID(**foreign key**) DislikeID (**foreign key**)

Likes
Attributes: LikesID NumberOfLikes

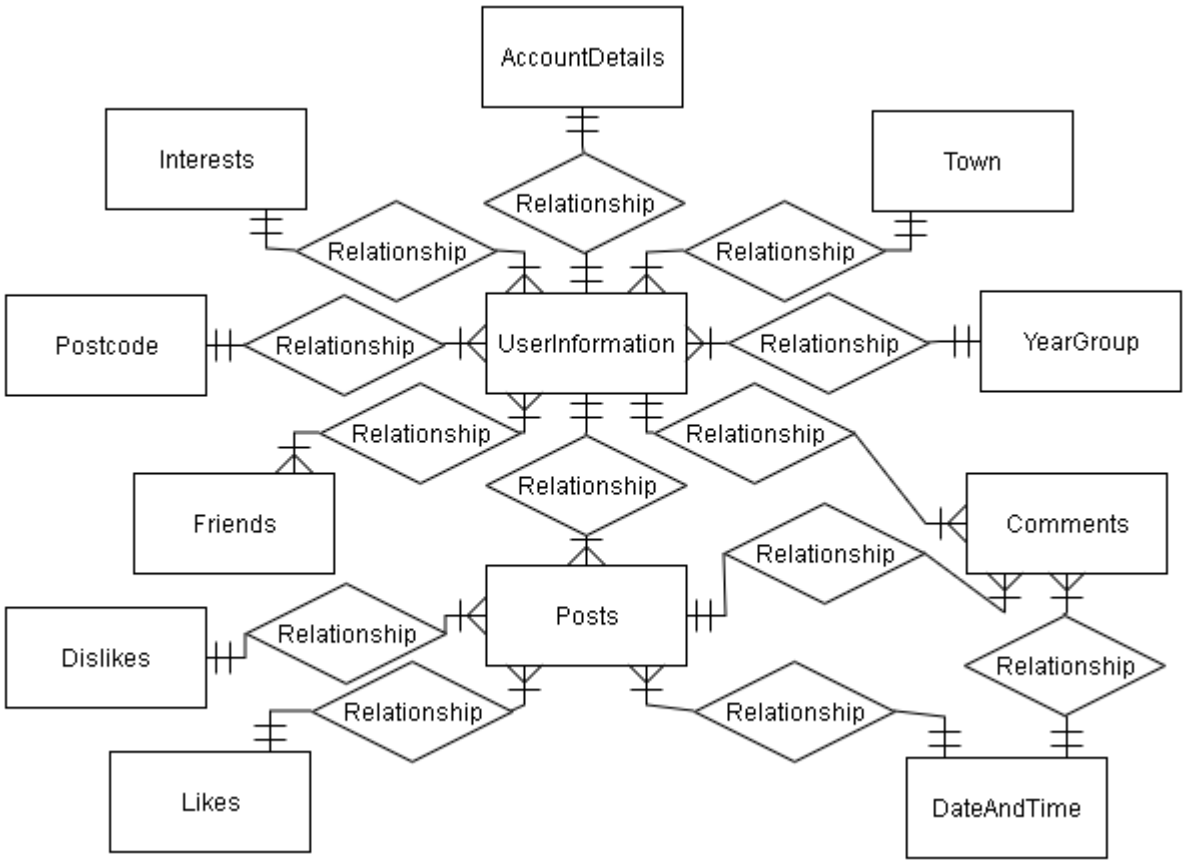
YearGroup
Attributes: YearID Year

DateAndTime
Attributes: TimeID Created

Dislikes
Attributes: DislikeID NumberOfDislikes

Entity Relationship Diagram[43]

This will display the relationships between each table graphically in third normal form. Double perpendicular bars on both ends of the relationship means the tables are linked in a one to one relationship. Bars with a > or < on both sides represent a many to many relationship but a relationship with only one > or < on the side of a relationship will show the relationship is a one to many relationship.



Data definition language

Pseudocode to create tables in the database:[44]

SQL	Purpose
CREATE TABLE IF NOT EXISTS accountdetails (**accountid INTEGER PRIMARY KEY,** **username CHAR(40) not null,** **password CHAR(40) not null,** **email CHAR(256) not null,** **reported BOOLEAN);**	This creates the account details table with accountid as an integer which will be the primary key, username and password with will be character fields in the database with length of 40 and can't be left null. Email will be a character field with a maximum of 256 characters and can't be left blank. Reported is a Boolean field that will initially be set to false.
CREATE TABLE IF NOT EXISTS year (**yearid INTEGER PRIMARY KEY,** **year INTEGER null);**	This creates the year table in the database with primary key yearid as an integer and year as an integer field that can be left blank.

[43] https://erdplus.com/#/
[44] http://www1.udel.edu/evelyn/SQL-Class1/SQLTable1.html

CREATE TABLE IF NOT EXISTS interests (**interestsid INTEGER PRIMARY KEY,** **interest CHAR(15) not null);**	This creates the interest table with interestid as an integer and primary key with the field interest as a character field with length of 15 characters that can't be left blank.
CREATE TABLE IF NOT EXISTS town (**townid INTEGER PRIMARY KEY,** **townname CHAR (40) not null);**	This creates the town table with townid as an integer and primary key with the field townname as a character field length of 40 characters that can't be left blank.
CREATE TABLE IF NOT EXISTS postcode (**postcodeid INTEGER PRIMARY KEY,** **postcode CHAR(7) not null);**	This creates the postcode table with postcodeid as an integer and primary key with the field postcode as a character field with length of 7 characters that can't be left blank.
CREATE TABLE IF NOT EXISTS userinformation (**userid INTEGER PRIMARY KEY,** **firstname CHAR(40) not null,** **lastname CHAR(40) not null,** **dateofbirth DATE not null,** **accountid INTEGER UNIQUE FOREIGN KEY REFERENCES accountdetails(accountid),** **yearid INTEGER FOREIGN KEY REFERENCES year(yearid),** **interestsid INTEGER FOREIGN KEY REFERENCES interest(interestid),** **postcodeid INTEGER FOREIGN KEY REFERENCES postcode(postcodeid),** **townid INTEGER FOREIGN KEY REFERENCES town(townid));**	This creates the userinformation table with userid as an integer primary key, first and last name as character fields with length of 40 characters which can't be left blank and date of birth as a date field that can't be left blank. This table will also link to the relevant tables associated with unique foreign keys signifying a one to one relationship and general foreign keys signifying one to many relationships with relevant reference to the fields which link each table to this table.
CREATE TABLE IF NOT EXISTS friends (**friendid INTEGER PRIMARY KEY,** **userid INTEGER FOREIGN KEY REFERENCES userinformation(userid));**	This creates the friends table with userid as a foreign key that is an integer and with a primary key freindid that is an integer.
CREATE TABLE IF NOT EXISTS comments (**commentid INTEGER PRIMARY KEY,** **userid INTEGER FOREIGN KEY REFERENCES userinformation(userid),** **postid INTEGER UNIQUE FOREIGN KEY REFERENCES posts(posttid),** **contentcommented CHAR(256) not null,** **timeid INTEGER FOREIGN KEY REFERENCES dateandtime(timeid));**	This creates the comments table with integer primary key as commentid, with unique foreign key to post and general foreign keys to dateandtime and userinformation with all of these foreign keys stored as integers.
CREATE TABLE IF NOT EXISTS posts (**postsid INTEGER PRIMARY KEY,** **userid INTEGER FOREIGN KEY REFERENCES userinformation(userid),** **title CHAR(256) not null,** **contentposted CHAR(256) not null,** **timeid INTEGER FOREIGN KEY REFERENCES dateandtime(timeid),** **likesid INTEGER FOREIGN KEY REFERENCES likes(likeid),**	This creates the posts table with postsid, userid, title, contentposted, timeid, likesid and dislikeid. All of the fields labelled with id as integer fields in the table. Postsid is the only id that is a primary key were as the other ids are foreign keys to their relevant tables. Title and contentposted are character fields of 256 characters that can't be left blank.

dislikeid INTEGER FOREIGN KEY REFERENCES dislikes(dislikeid));	
CREATE TABLE IF NOT EXISTS likes (likesid INTEGER PRIMARY KEY, numberoflikes INTEGER null);	This creates the table for likes with primary key as an integer for likesid and number of likes as an integer with initial value set to null.
CREATE TABLE dislikes (dislikeid INTEGER PRIMARY KEY, numberofdislikes INTEGER null);	This creates the table for dislikes with primary key as an integer for dislikeid and number of dislikes as an integer with initial values set to null.
CREATE TABLE IF NOT EXISTS dateandtime (timeID INTEGER PRIMARY KEY, created DATE not null);	This creates the table for date and time with primary key as an integer for timeid and created as a date field which can't be left blank.

Pseudocode for insert statement:[45]

SQL	PURPOSE
INSERT ("%s", "%s", "%s"), INTO ("%s" = accountdetails.email, "%s" = accountdetails.username, "%s" = accountdetails.password), FROM (accountdetails);	This inserts the username, email and password that the user inputs into the form into their relevant places in the database table for accountdetails.
INSERT ("%s", "%s,", "%d", "%d", "%s", "%s", "%s"), INTO(userinformation.firstname, userinformation.lastname, userinformation.dateofbirth, year.year, interests.interest, postcode.postcode, town.town), FROM(accountdetails, userinformation, year, interests, postcode, town), WHERE (userinformation.accountid = accountdetails.accountid, userinformation.yearid = year.yearid, userinformation.interestid = interests.interestid, userinformation.postcodeid = postcode.postcodeid, userinformation.townid = town.townid);	This inserts the information completed in the user information form into the database assigned to each respective table and by linking the appropriate foreign keys with the tables that the information is being called from.
INSERT ("%s", "%s", "%s"), INTO(posts.title, posts.content, posts.image), FROM(posts) WHERE(userinformation.userid = post.userid);	This inserts the title, content and file directory for the image as a string in the database table for posts when a user creates a post which will assign the post to the relevant user.
INSERT ("%s"), INTO (comments.contentcommented), FROM(comments), WHERE(posts.postid = comments.postid, Userinformation.userid=comments.userid);	This inserts the comment the user has into the relevant field in the comments database table which is assigned to the post that the comment has been made on and the user who commented the content.
INSERT("%d", "%d"), INTO (friends.userid), FROM (friends);	This inserts the users' primary keys into the friends table creating a relationship between the two users.

[45] https://www.techonthenet.com/sql/insert_ddl.php

Pseudocode for select statements:

SQL	Purpose
SELECT (userinformation.yearid, userinformation.interestid, userinformation.townid, userinformation.postcodeid, userinformation.firstname, userinformaiton.lastname, userinformation.dateofbirth), FROM (userinformation), WHERE (userinformation.accountid = accountdetails.accountid);	This will select all the profile data from the database for each user
SELECT (friends.friendid), FROM (friends), WHERE (friends.userid = userinformation.userid);	This will select all the friends that a user has from the database.
SELECT (post.title, post.image, post.content), FROM (posts), WHERE (post.userid = userinformation.userid), ORDER BY(posts.timeid);	This will select all the posts a user has made and order the posts by the time in which they were created.

Pseudocode for update statements

SQL	Purpose
UPDATE (userinformation), SET (userinformaiton.yearid = %d, userinformation.interestid = %d, userinformation.townid = %d, userinformation.postcodeid = %d), WHERE (userinformation.accountid = accountdetails.accountid);	This will update the profile of each user so that users are able to change their year group, interests, postcode or town that they live in.

Banned Words algorithm[46]

This will be the pseudocode that I will use for detecting if any banned words are used in any forms accessed on the social network.

Pseudocode for sign up page

```
var username = Textbox.username
var firstname = Textbox.firstname
var lastname = Textbox.lastname
var email = Textbox.email
load the banned words from the text file and store it into var bannedwords
FOR each word in bannedwords THEN
        IF word not in username and firstname and email and lastname THEN
                Login Allowed
                Show Menu screen
```

[46] https://study.com/academy/lesson/writing-pseudocode-algorithms-examples.html

```
        ELSE
            Show error message
        ENDIF
ENDFOR
```

Pseudocode for profile

```
var postcode = Textbox.postcode
var town = Textbox.town
load the banned words from the text file and store it into var bannedwords
FOR each word in bannedwords THEN
        IF word not in town and postcode THEN
                Login Allowed
                Show Menu screen
        ELSE
            Show error message
        ENDIF
ENDFOR
```

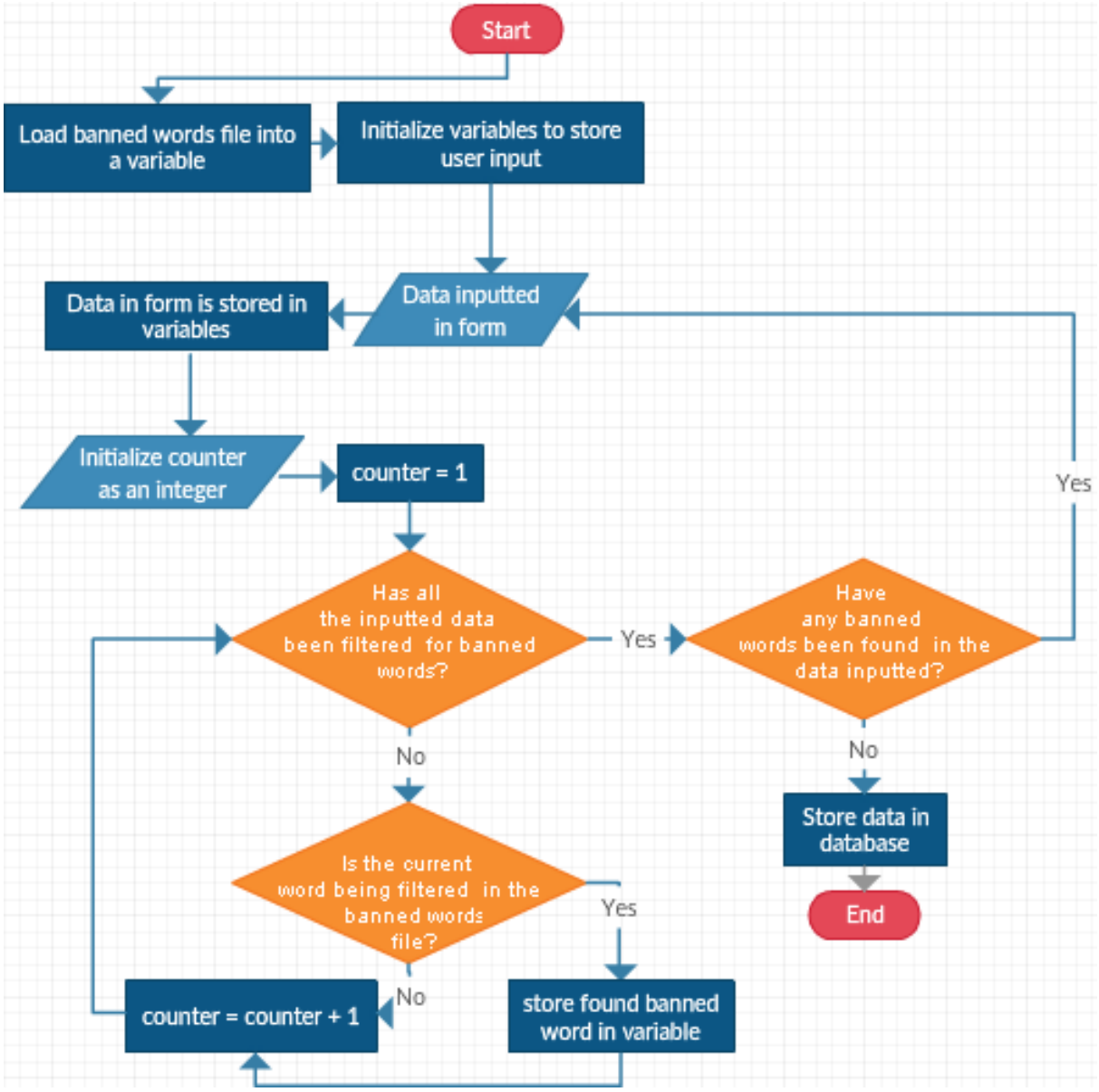
Pseudocode for posts

```
var title = Textbox.title
var content = Textbox.content
load the banned words from the text file and store it into var bannedwords
FOR each word in bannedwords THEN
        IF word not in title and content THEN
                Login Allowed
                Show Menu screen
        ELSE
            Show error message
        ENDIF
ENDFOR
```

Pseudocode for comments

```
var comment = Textbox.commentcommented
load the banned words from the text file and store it into var bannedwords
FOR each word in bannedwords THEN
        IF word not in comment THEN
                Login Allowed
                Show Menu screen
        ELSE
            Show error message
        ENDIF
ENDFOR
```

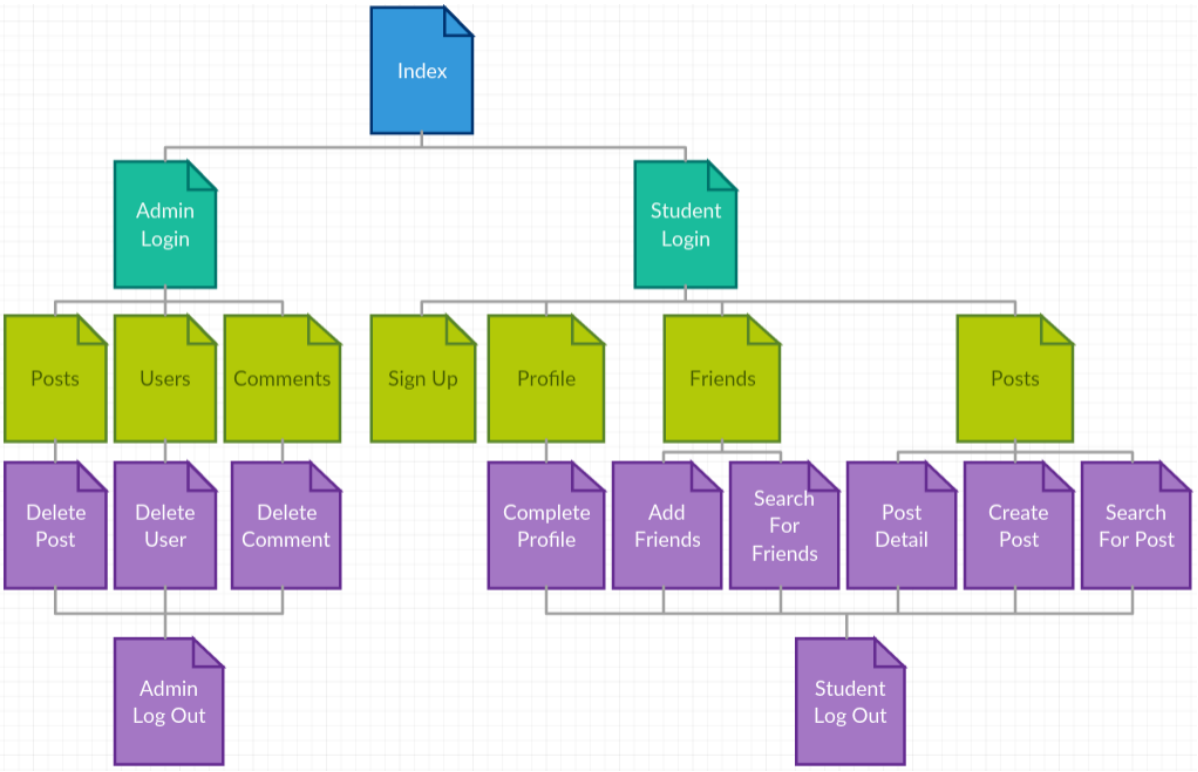
Flowchart



This is the main structure of how the banned words algorithm should work detailed in flowchart format.

TECHNICAL SOLUTION

Actual webpage navigation



These pages are all implemented using html (Hyper Text Markup Language). Although logout is connected to complete profile, add friends, search for friends, post detail, search for post and create post, logout is also connected to profile, friends and post. Additionally, profile, friends and posts are all connected therefore you are able to access profile from friends and posts and vice versa. To avoid causing confusion I have displayed the webpage navigation like this so that a user is able to understand the hierarchical structure of the web application.

Technology used

Django[47]

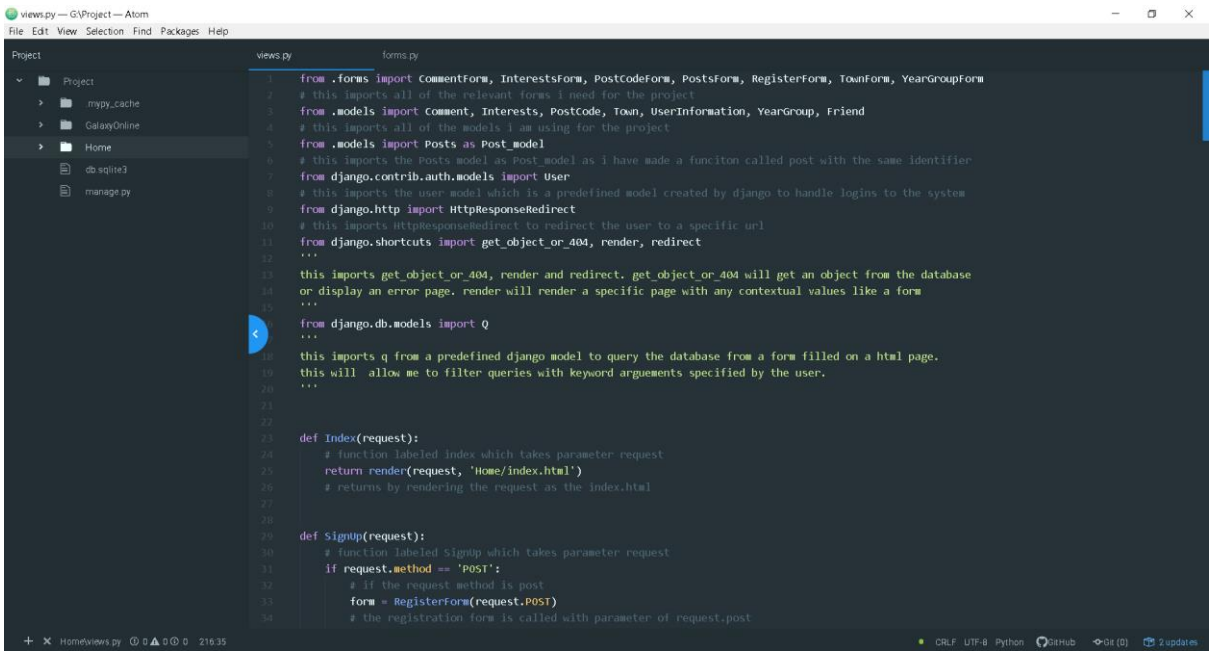
In terms of the technology I have used, I have used Django to use a python backend to act as a handler for the forms and validation on the website to replace the need for JavaScript as I don't have the required knowledge to use JavaScript correctly.

Django will also allow me to run a development server so that I am able to access the webpage and forms on the web application to view if they are functioning correctly and will allow me to interact with the website which will help me improve how the website functions or looks. Additionally, due to the file structure of a Django project I will be able to edit or review pieces of code that I have created easily and I will be able to look for guidance for developing my project on the internet.

[47] https://www.djangoproject.com/start/overview/

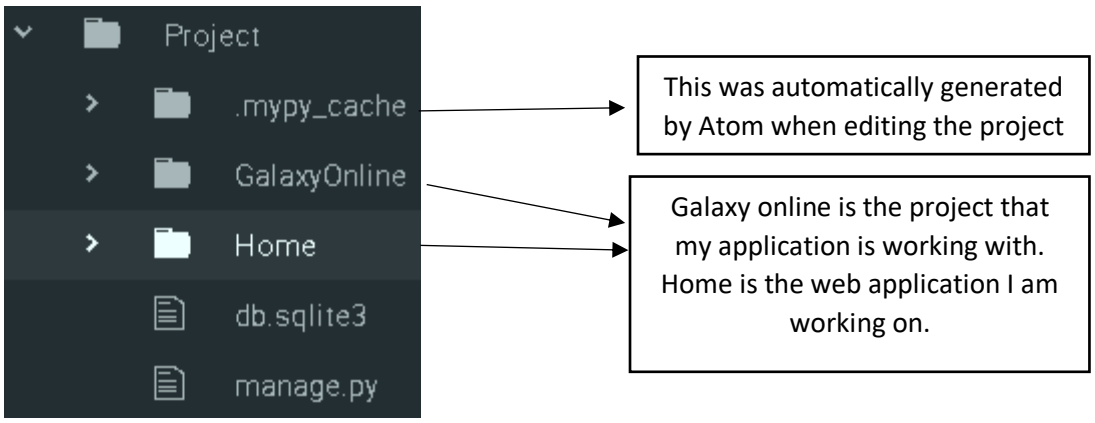
Atom[48]

To code my project I have used the text editor called atom which I can also use as an integrated development environment for python and Django.

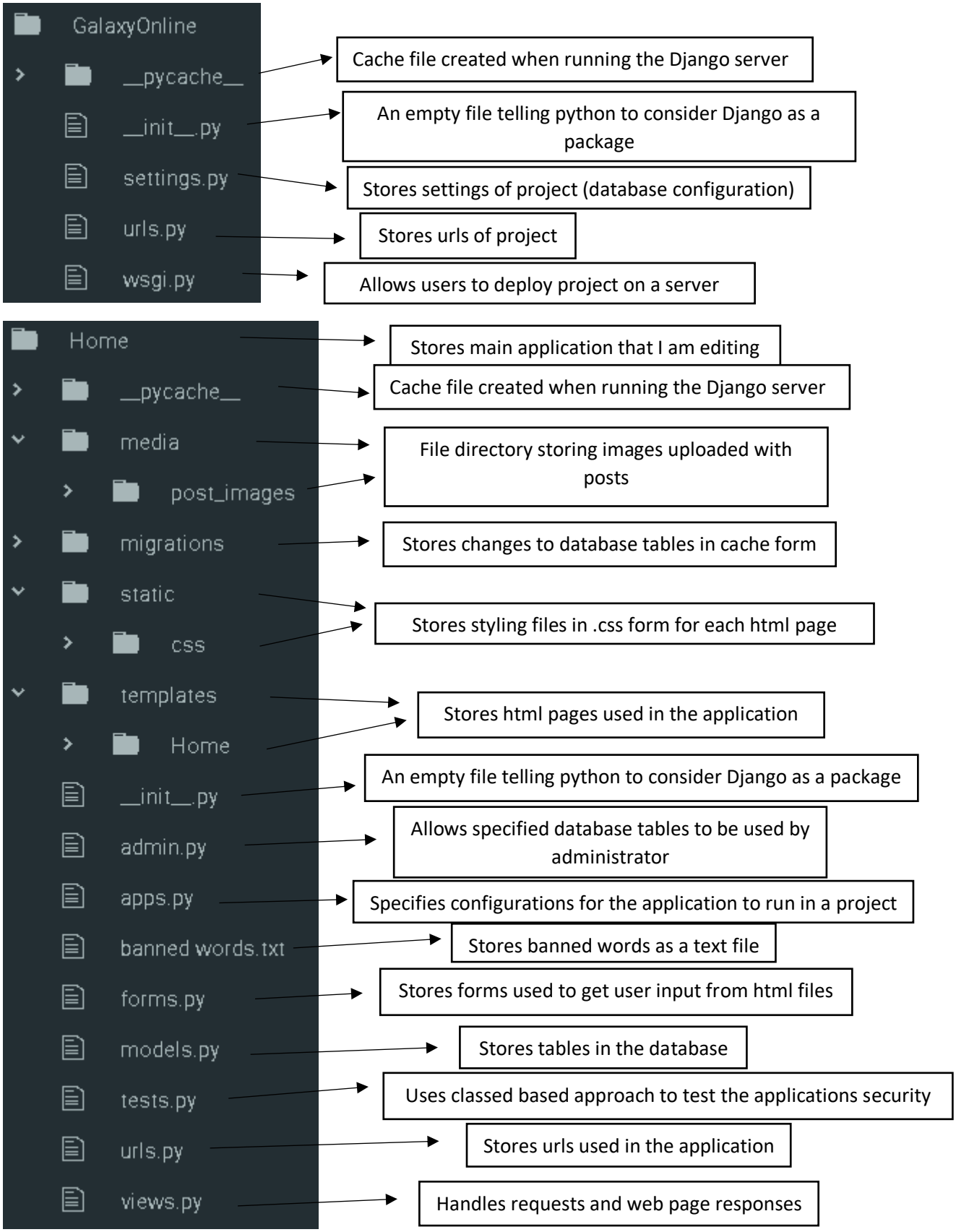


Atom allows users to install packages which I have taken the liberty to install packages such as terminal to run the development server for Django and python linters and syntax highlighters when editing my code. Additionally, due to the file structure being displayed next to the code, switching between the files is very simple and efficient.

File structure



[48] https://atom.io/


GalaxyOnline
__pycache__
__init__.py
settings.py
urls.py
wsgi.py
Cache file created when running the Django server
An empty file telling python to consider Django as a package
Stores settings of project (database configuration)
Stores urls of project
Allows users to deploy project on a server
Home
__pycache__
media
post_images
migrations
static
css
templates
Home
__init__.py
admin.py
apps.py
banned words.txt
forms.py
models.py
tests.py
urls.py
views.py
Stores main application that I am editing
Cache file created when running the Django server
File directory storing images uploaded with posts
Stores changes to database tables in cache form
Stores styling files in .css form for each html page
Stores html pages used in the application
An empty file telling python to consider Django as a package
Allows specified database tables to be used by administrator
Specifies configurations for the application to run in a project
Stores banned words as a text file
Stores forms used to get user input from html files
Stores tables in the database
Uses classed based approach to test the applications security
Stores urls used in the application
Handles requests and web page responses

Code I've added:

In settings.py[49]

```
STATIC_URL = '/static/'
# this was automatically generated by django
# this stores the static files in a static directory

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'Home/media')
# this stores the media files in a media directory

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]
# i added this to the file as to allow the project to locate my static folder

LOGIN_REDIRECT_URL = 'Profile'
# this redirects the user to the profile page after loggin into the system
```

In GalaxyOnline/urls.py

```
from django.contrib import admin
# imports admin to use for urls
from django.urls import include, path
# imports include and path to use for mapping urls
from django.conf import settings
# imports settings from settings.py
from django.conf.urls.static import static
# imports static to store/load static files from html forms

urlpatterns = [
    path('', include('Home.urls')),
    path('admin/', admin.site.urls),
    # this allows my apps urls and the admin urls to be used
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
# added static line from django docs and allows images to be stored in MEDIA_ROOT
```

Imports in Home/forms.py

```
from django import forms
# imports form method
from django.contrib.auth.models import User
# imports user table from database
from django.contrib.auth.forms import UserCreationForm
# imports UserCreationForm to use when users sign up
from .models import YearGroup, Interests, PostCode, Town, Posts, Comment
# imports database tables from models.py
```

[49] https://docs.djangoproject.com/en/2.1/howto/static-files/

Imports in Home/models.py

```
from django.contrib.auth.models import User
# imports user table automatically generated by Django
from django.db import models
# imports models for me to use for my database tables
from django.db.models.signals import post_save
# imports post_save to be used after a record has been saved in a table
from django.dispatch import receiver
# can be used to connect signals after a record has been saved in a table
from django.urls import reverse
# can be used to reverse url arguements connected to a database table
# Create your models here.
```

Imports in Home/urls.py

```
from . import views
# imports all of the functions i have created in views
from django.contrib.auth import views as auth_views
# imports authorisation views
from django.urls import path
# imports path to link to urls that html files are connected to
# everything aove this line was autogenerated by django
```

Imports in Home/views.py

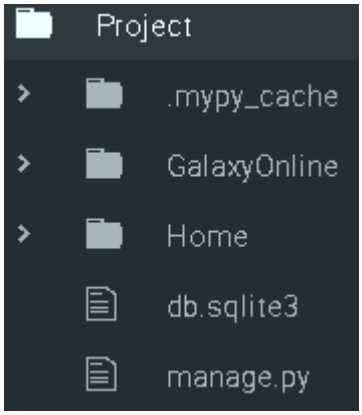
```
from .forms import CommentForm, InterestsForm, PostCodeForm, PostsForm, RegisterForm, TownForm, YearGroupForm
# this imports all of the relevant forms i need for the project
from .models import Comment, Interests, PostCode, Town, UserInformation, YearGroup, Friend
# this imports all of the models i am using for the project
from .models import Posts as Post_model
# this imports the Posts model as Post_model as i have made a funciton called post with the same identifier
from django.contrib.auth.models import User
# this imports the user model which is a predefined model created by django to handle logins to the system
from django.http import HttpResponseRedirect
# this imports HttpResponseRedirect to redirect the user to a specific url
from django.shortcuts import get_object_or_404, render, redirect
'''
this imports get_object_or_404, render and redirect. get_object_or_404 will get an object from the database
or display an error page. render will render a specific page with any contextual values like a form
'''
from django.db.models import Q
'''
this imports q from a predefined django model to query the database from a form filled on a html page.
this will  allow me to filter queries with keyword arguements specified by the user.
'''
```

Server

A client server model is a structure that allows users to interact with an application of a piece of software from a client to a server. In my program, I will be using a client server model to allow users to be able to access the social network from any client device and will be able to interact with the application hosted on a server or host computer. Within my coursework, I will be using my computer to act as a server to host the web application for users to access from local clients.

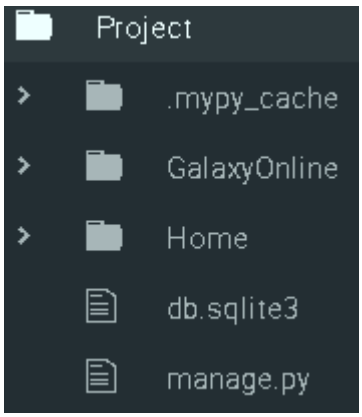
I have decided to use a client server model as it will allow many students to be able to use my software without needing high demanding hardware requirements. Through a client server model,

using a computer with at least 1 GB of ram will allow users to access the internet to interact with my web application and will mean that interaction with the project will not be limited by the hardware that a computer has access to but the speed of a user's internet connection.



To run the web application manage.py is run via command line on the host computer. If the host computer has Django installed a development server will be run allowing users to access the web application but if the Django module is not installed, an error is raised.

Database[50]



The file used to store my database tables is db.sqlite3

Django uses web APIs to carry out SQL queries on the database. The Home/models.py file is used to create or edit the tables in the database such as relationships or fields in the database. The db.sqlite3 file stores the file in sql format so that the file can be viewed in a SQL supported browser such as DB Browser for SQLite.

For this project there are 8 database tables. Although this differs from my design of the database, the functionality of the system is unaffected. The tables which are in the database which I have created are Comment, Interests, PostCode, Town, UserInformation, YearGroup, Friend, Posts and User. Upon creating the database in Django, I realised that it was not necessary to have separate tables in the database for the date and time a comment or post is published and that the implementation of having likes and dislikes in the database as separate tables was unnecessary. Although Django automatically creates the User table, I have used it to store the email, first name, last name, username and passwords of each user and have used it to handle the login system to the social network.

[50] https://docs.djangoproject.com/en/2.1/topics/db/models/

Comment

In models.py[51]

```
class Comment(models.Model):
    # create table in database
    post = models.ForeignKey(Posts, on_delete=models.CASCADE)
    # ManyToOne field to Posts
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    # ManyToOne field to Users
    content = models.TextField(max_length=256)
    # content as text field with max length of 256 characters
    publish = models.DateTimeField(auto_now_add=True)
    # publish as a date time field storing the time a comment is made

    def __str__(self):
        return "{}-{}".format(self.post.title, str(self.user.username))
        # returns string value of the posts title and users username in table
```

SQL

CREATE TABLE "Home_comment" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "content" text NOT NULL, "publish" datetime NOT NULL, "post_id" integer NOT NULL REFERENCES "Home_posts" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED)

Interests

In models.py

```
class Interests(models.Model):
    # creates interest table in database
    choice = (("Music", "Music"), ("Reading", "Reading"), ("Tv or Movies", "Tv or Movies"), ("Video Games", "Video Games"), ("Art", "Art"),)
    # stores choices that can be made for interest
    interest = models.CharField(max_length=15, choices=choice)
    # sets interest as a character field with max length of 15 characters with choices specified
```

SQL

CREATE TABLE "Home_interests" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "interest" varchar(15) NOT NULL)

PostCode

In models.py

```
class PostCode(models.Model):
    # creates postcode table in database
    postcode = models.CharField(max_length=7)
    # sets postcode as a character field with max length of 7 characters
```

[51] https://youtu.be/An4hW4TjKhE

SQL

CREATE TABLE "Home_postcode" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "postcode" varchar(7) NOT NULL)

Town

In models.py

```
class Town(models.Model):
    # creates town table in database
    town = models.CharField(max_length=40)
    # sets town as a character field with max length of 40 characters
```

SQL

CREATE TABLE "Home_town" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "town" varchar(40) NOT NULL)

UserInformation

In models.py

```
class UserInformation(models.Model):
    # creates userinformation table in database
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    # OneToOne relationship with user table
    year = models.ForeignKey(YearGroup, on_delete=models.PROTECT, null=True)
    # ManyToOne relationship with yeargroup table, value can be blank
    interest = models.ForeignKey(Interests, on_delete=models.PROTECT, null=True)
    # ManyToOne relationship with interests table, value can be blank
    postcode = models.ForeignKey(PostCode, on_delete=models.PROTECT, null=True)
    # ManyToOne relationship with postcode table, value can be blank
    town = models.ForeignKey(Town, on_delete=models.PROTECT, null=True)
    # ManyToOne relationship with town table, value can be blank
```

SQL

CREATE TABLE "Home_userinformation" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "interest_id" integer NULL REFERENCES "Home_interests" ("id") DEFERRABLE INITIALLY DEFERRED, "postcode_id" integer NULL REFERENCES "Home_postcode" ("id") DEFERRABLE INITIALLY DEFERRED, "town_id" integer NULL REFERENCES "Home_town" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL UNIQUE REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "year_id" integer NULL REFERENCES "Home_yeargroup" ("id") DEFERRABLE INITIALLY DEFERRED)

YearGroup

In models.py

```
class YearGroup(models.Model):
    # creates yeargroup table
    year = models.IntegerField(null=True)
    # sets year as a field in the table as an integer field which can be left blank
```

SQL

CREATE TABLE "Home_yeargroup" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "year" integer NULL)

Friend

In models.py[52]

```
class Friend(models.Model):
    # create friend table in database
    users = models.ManyToManyField(User)
    # manytomany relationship with user table
    current_user = models.ForeignKey(User, related_name='owner', on_delete=models.CASCADE, null=True)
    # ManyToOne relationship with user table as owner of friendship
```

SQL

CREATE TABLE "Home_friend" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "current_user_id" integer NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED)

CREATE TABLE "Home_friend_users" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "friend_id" integer NOT NULL REFERENCES "Home_friend" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED)

[52] https://github.com/maxg203/Django-Tutorials/blob/master/home/models.py

Posts

In models.py[53]

```
class Posts(models.Model):
    # create posts table in database
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    # ManyToOne relationship with user
    title = models.CharField(max_length=200)
    # title is a charfield with max length of 200 characters
    content = models.TextField()
    # content is a text field with no max length
    image = models.ImageField(upload_to='post_images', blank=True, null=True)
    # installed pillow to use image field
    # image is a image field which will upload an image to post_images
    # this field can be left blank
    publish = models.DateTimeField(auto_now_add=True, auto_now=False)
    # publish is a datetimefield that sets the time as soon as the post is saved to the database
    likes = models.ManyToManyField(User, related_name="likes", blank=True)
    # likes is a manytomany relationship with user that can be left blank
```

SQL

CREATE TABLE "Home_posts" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(200) NOT NULL, "content" text NOT NULL, "image" varchar(100) NULL, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "publish" datetime NOT NULL)

CREATE TABLE "Home_posts_likes" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "posts_id" integer NOT NULL REFERENCES "Home_posts" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED)

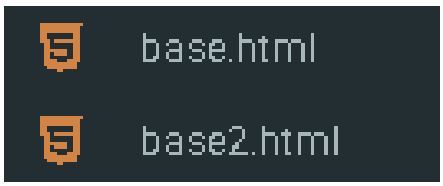
Relationships

Foreign key fields represent many to one relationships. One to one fields represent one to one relationships. Many to Many fields represent many to many relationships. Within each one to one relationship or foreign key field I have specified an on_delete parameter. This parameter is a form of association in Django as the database tables are written as classes therefore interactions with records in the database are treated as objects or instances of the classes within Django. When using models.CASCADE in the on_delete parameter, this is a form of composition which means that when the object or instance of that class is deleted, the related object will no longer exist. For example, if a user were removed from the database the posts a user makes would be deleted but if a user's posts are deleted from the database. However, I have also used models.PROTECT which is a form of aggregation which means that when or object or instance of a class is deleted, the related objects will still exist. For example, if a user were to delete a post on their account their account would not be deleted.

[53] https://youtu.be/2h57cqFRcqg

Pages[54]

Base Pages



These html files will store the html structure of each page on the website thus storing the title, head and body tags on the page.

HTML for base.html

```
<!DOCTYPE html>
{#defines that this is a html document#}
{% load static %}
{#django funciton to load css styles for page#}
<html lang="en">
{#defines that the html document is in english#}
<head>
    {#head tag for html document#}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    {#above meta tags auto generated for html document#}
    <link rel="stylesheet" type="text/css" href="{% static 'css/base.css' %}">
    {#link to the css file to load the style for this page#}
    {% block css%}{% endblock %}
    {#link to css block from child pages#}
    <title>{% block title %}{% endblock %}</title>
    {#title tag to store title of page, block title used to link to title on child pages#}
    {#end tag for title block and title ends the space for the given variable on child page#}
</head>
{#end tag for head element#}
<body>
    {#starting body tage#}
    {% block body %}{% endblock %}
    {#blocks for body to link to body elements in child page, End block to show end of body block on child page#}
</body>
{#end of body tag to show end of body element on page#}
</html>
{#end of html tag to show end of html document#}
```

[54] https://tutorial.djangogirls.org/en/template_extending/

CSS to style base.html in base.css

```css
body{
    background-color: midnightblue;
    /* this makes the background of the body element midnightblue */
}
ul{
    list-style-type: none;
    color: red;
    /*this makes the form errors on the page have no bullet points but appear red*/
}
p{
    color: white;
    /*this makes the paragraph have white colored text*/
    font-size: 12px
    /*this makes the paragraph have a font size of 12 pixels*/
}
```

HTML for base2.html

```
<!DOCTYPE html>
{#defines that this is a html document#}
{% load static %}
{#django funciton to load css styles for page#}
<html lang="en">
{#defines that the html document is in english#}
<head>
    {#head tag for html document#}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    {#above meta tags auto generated for html document#}
    <link rel="stylesheet" type="text/css" href="{% static 'css/base2.css' %}">
    {#link to the css file to load the style for this page#}
    {% block css%}{% endblock %}
    {#link to css block from child pages#}
    <title>{% block title %}{% endblock %}</title>
    {#title tag to store title of page. block title used to link to title on child pages#}
    {#end tag for title block and title ends the space for the given variable on child page#}
</head>
{#end tag for head element#}
<body>
    {#starting body tage#}
    {% block body %}{% endblock %}
    {#blocks for body to link to body elements in child page. End block to show end of body block on child page#}
</body>
{#end of body tag to show end of body element on page#}
</html>
{#end of html tag to show end of html document#}
```

CSS styling for base2.html in base2.css

```
body{
    background-color: #368BC1;
    /* this makes the background of the body element lightblue */
}
ul{
    list-style-type: none;
    color: black;
    /*this makes form errors appear black*/
}
p{
    color: black;
    /*this makes the paragraph have black colored text*/
    font-size: 25px
    /*this makes the paragraph have a font size of 25 pixels*/
}
.Button1{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:7.5vw;
    /*make font size of button 7.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: green;
    /*have background of button set to green*/
    border:black;
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
```

```
body{
    background-color: #368BC1;
    /* this makes the background of the body element lightblue */
}
ul{
    list-style-type: none;
    color: black;
    /*this makes form errors appear black*/
}
p{
    color: black;
    /*this makes the paragraph have black colored text*/
    font-size: 25px
    /*this makes the paragraph have a font size of 25 pixels*/
}
.Button1{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:7.5vw;
    /*make font size of button 7.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: green;
    /*have background of button set to green*/
    border:black;
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
```

```
.Button1:hover{
    background-color: lightgreen
    /*have button set to lightgreen when hovering over button*/
}
.Button1:active{
    background-color:lightgreen;
    /*when button clicked changed color to lightgreen*/
    transform:translateY(2px)
    /*move button down 2 pixels*/
}
.Button2{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:7.5vw;
    /*make font size of button 7.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: blue;
    /*have background of button set to blue*/
    border:black;
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
.Button2:hover{
    background-color: lightblue
    /*have button set to lightblue when hovering over button*/
}
```

```
.Button2:active{
    background-color:lightblue;
    /*when button clicked changed color to lightblue*/
    transform:translateY(2px)
    /*move button down 2 pixels*/
}
.Button3{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:7.5vw;
    /*make font size of button 7.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: red;
    /*have background of button set to red*/
    border:black;
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
.Button3:hover{
    background-color: rgb(220,20, 60)
    /*have button set to lightred when hovering over button*/
}
.Button3:active{
    background-color: rgb(220,20, 60);
    /*when button clicked changed color to lightred*/
    transform:translateY(2px)
    /*move button down 2 pixels*/
}
```

```
.container {
    margin: auto;
    /* this makes the margin of the container automatically adjust */
    align: center
}
```

Index page

This page redirects users to the admin login page or the student login page.

HTML in index.html

```
{% extends "Home/base.html" %}
{#extends the base.html page. This shows the relationship with the parent page base.html and this child page#}
{% load static %}
{#load css file to use for styling#}
{% block css %}<link rel="stylesheet" type="text/css" href="{% static 'css/index.css' %}">{% endblock %}
{#load css file as a link that can be read as a text or css file with reference to file location#}
{#block css used to link to block element in parent page to this page#}
{% block body %}
{#start of body elements#}
    <div class= "Container">
        {# create div class called container #}
        <h1 align="center">{% block title %}Galaxy Online{% endblock %}</h1>
        {#open tag for heading to be aligned in center of page. opening block title used to show that#}
        {#variable between block and endblock is used for title in parent page. endblock shows there is no more#}
        {#content for the given variable. End tag for heading to show end of heading#}
        <div id = "Buttons" align="center">
            {# create div with id of buttons and align the div to center of page #}
            <table border = "0">
                {# create table with border that is not visible #}
                <tr>
                {%block content%}
                {%endblock%}
                    {# create first row for table #}
                    <td>
                        {# create first column for table #}
                        <a class="button" href = "{% url 'admin:index' %}">Admin</a>
                        {#create a button that links to the admin page. the url admin:index#}
                        {#is used to identify the admin login page as it isnt a physical file#}
                        {#opening and closing of the "a" tag shows the start and end of the linked button#}
                    </td>
                    {# end tag for column of table #}
                    <td></td>
                    {# create blank column for table #}
                    <td>
                        {#open tag for column of table#}
                        <a class="button" href = "{% url 'Login' %}">Student</a>
                        {#create a button that links to the login page. the url login#}
                        {#links the button to the function in view called Login to display#}
                        {#the relevant page. the opening and closing of the "a" tag shows the start and end#}
```

```
                        {#of the linked button#}
                    </td>
                    {#end tag for column of table#}
                </tr>
                {#end tag for row of table#}
            </table>
            {#end tag for table#}
        </div>
        {#end tag for div#}
    </div>
    {#end tag for div#}
{% endblock %}
{#end of body elements#}
```

Therefore if the admin button is clicked, the admin login page is displayed. If the student button is clicked, the student login page is displayed.

CSS in index.css

```css
.Container {
    border: 1px solid black;
    /* this makes the container have a border of 1 pixel that is black */
    background-color: darkblue;
    /* this makes the container have a background colour of darkblue */
    margin: auto;
    /* this makes the margin of the container automatically adjust */
    margin-top: 120px;
    /* this makes the containers top margin from the page persistantly 120 pixels */
    width: 640px
    /* this makes the width of the container 640px */
}
h1{
    font-size:6vw;
    /* this makes the heading have a font size that is 6% of the viewport width */
    color: darkturquoise
    /* this makes the heading have a darkturquoise color */
}
.button{
    display:inline-block;
    /* this lets us set a width and height to the button */
    padding:auto;
    /* this makes the padding around the button automatic */
    font-size:3.5vw;
    /* this makes the font size in the button 3.5% of the viewport width*/
    cursor:pointer;
    /* this changes the cursor to a pointer when hovering over button*/
    text-align: center;
    /* this has the text in the button align in the center of the button*/
    outline:black;
    /* this makes the outline of the text black*/
    color: black;
    /* this makes the text color black*/
    background-color: grey;
    /* this makes the background of the button grey */
    border:black;
    /*this makes the border of the button black*/
    border-radius: 3px;
    /*this gives a rounded effect to the button of 3 pixels*/
```

```
}
.button:hover{
    background-color: lightgrey
    /*this makes the background colour of the button lightgrey when the cursor hovers over the button*/
}
.button:active{
    background-color:lightgrey;
    /* this maes the background color lightgrey when the button is clicked*/
    transform:translateY(2px)
    /*this moves the button down by 2 pixels when clicked*/
}
@media screen and (max-width: 640px) {
    /*when the page screen has a width less than or equal to 640 pixels*/
    .Container{
        width: 100%;
        /*make the width of the container the size of the page*/
    }
}
@media screen and (max-height: 480px) {
    /*when the page screen has a height less than or equal to 480 pixels*/
    .Container{
        margin: auto;
        /*make the margin of the container automatically adjust*/
    }
}
```

In Home/views.py

```
def Index(request):
    # function labeled index which takes parameter request
    return render(request, 'Home/index.html')
    # returns by rendering the request as the index.html
```

This renders the page via the render method imported in views.py.

In Home/urls.py

```
path('', views.Index, name='index'),
```

This links the index function to the URL for the index page.

Student Sign in[55]

This page allows users to either login to the social network or be redirected to the signup page.

[55] https://simpleisbetterthancomplex.com/tutorial/2016/06/27/how-to-use-djangos-built-in-login-system.html

HTML in Student.html

```
{% extends "Home/base.html" %}
{#extends the base.html page. This shows the relationship with the parent page base.html and this child page#}
{% load static %}
{#load css file to use for styling#}
{% block css %}<link rel="stylesheet" type="text/css" href="{% static 'css/Student.css' %}">{% endblock %}
{#load css file as a link that can be read as a text or css file with reference to file location#}
{#block css used to link to block element in parent page to this page#}
{% block body %}
{#start of body elements#}
    <div class="Container">
        {# create div class called container #}
        <h1 align="center">{% block title %}Student Sign In{% endblock %}</h1>
        {#open tag for heading to be aligned in center of page. opening block title used to show that#}
        {#variable between block and endblock is used for title in parent page. endblock shows there is no more#}
        {#content for the given variable. End tag for heading to show end of heading#}
        <div class="LoginButtons" align="center">
            {#create div class called loginbuttons that is aligned to the center of the page#}
            <form method = 'post'>
                {#start of form#}
                {% csrf_token %}
                {#this is a csrf_token which securely collects the information submitted in the form#}
                <table border="0">
                    {#start of table with invisible border#}
                    <tr>
                        <p>
                        {% for field in form%}
                            <br>{{field.label_tag}}
                            <br>{{ field }}
                        {% endfor %}
                        </p>
                        {% for error in form.non_field_errors %}
                            <ul>{{error}}</ul>
                        {% endfor %}
                        {#this displays all of the errors that will display in the form#}
                        <td>
                            {#column tag#}
                        <button type="submit" class="LoginButton">Login</button>
                        {#create login button#}
                        </td>
```

```
                        {#end of column#}
                    </tr>
                    {#end of row#}
                    <tr>
                        {#start of new row#}
                        <td>
                            {#new column#}
                        <a href="{% url 'SignUp' %}" class="SignUpButton">Sign Up</button>
                            {#linked button to url SignUp in view which displays SignUp.html#}
                        </td>
                        {#end of column#}
                    </tr>
                    {#end of row#}
                </table>
                {#end of table#}
            </form>
            {#end of form#}
        </div>
        {#end of div#}
    </div>
    {#end of div#}
{% endblock %}
{#end of body block#}
```

If login is clicked, the credentials of the user is checked with the credentials of users in the user database table. If the credentials match then the user is redirected to their profile page. If the credentials do not match, errors are displayed in the form. If the sign up button is clicked, the user is redirected to the signup page.

CSS in Student.css

```
h1{
    color:white;
    /*make heading color white*/
    font-size:5vw
    /*make font size of heading 5% of the viewport width*/
}
ul{
    color: red;
    /*this makes the form ValidationError appear red*/
}
.Container{
    border: 1px solid black;
    /*give container a black border that is 1 pixel thick*/
    background-color: darkblue;
    /*make background of container darkblue*/
    margin: auto;
    /*make margin of container automatically adjustable*/
    margin-top: 120px;
    /*make margin from top of page 120 pixels*/
    width: 640px
    /*make width of container 640 pixels*/
}
.LoginButton{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:3.5vw;
    /*make font size of button 3.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: green;
    /*have background of button set to green*/
    border:black;
```

```
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
.LoginButton:hover{
    background-color: lightgreen
    /*have button set to lightgreen when hovering over button*/
}
.LoginButton:active{
    background-color:lightgreen;
    /*when button clicked changed color to lightgreen*/
    transform:translateY(2px)
    /*move button down 2 pixels*/
}
.SignUpButton{
    display:inline-block;
    /*allow for adjusting height and width values*/
    font-size:3.5vw;
    /*set font size of button to 3.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*have outline of text as black*/
    color: white;
    /*have color of text as white*/
    background-color: red;
    /*have background of button as red*/
    border:black;
    /*have border of button set to black*/
    border-radius: 3px;
    /*have border of button be 3 pixels and rounded*/
}
.SignUpButton:hover{
    background-color: rgb(220,20, 60)
    /*on hover change button to this color*/
}
.SignUpButton:active{
```

```
    background-color: rgb(220,20, 60);
    /*on click change button to this color*/
    transform:translateY(2px)
    /*on click move button down by 2 pixels*/
}
@media screen and (max-width: 640px) {
    /*when width of page is 640 pixels or less*/
    .Container{
        width: 100%;
        /*make this container the same width as page*/
    }
}
@media screen and (max-height: 480px) {
    /*when height of page is 480 pixels or less*/
    .Container{
        margin: auto;
        /*make container margin automatically adjust*/
    }
}
```

In Home/urls.py

```
path('Login', auth_views.LoginView.as_view(template_name='Home/Student.html'), name='Login'),
```

This assigns the Student.html file to be the student login page by setting the auth_views.LoginView as the Student.html template.

Sign up[56]

This page allows users to fill out their signup details and submit their information to the database.

[56] https://www.youtube.com/watch?v=66l9b2QrBR8&list=PLw02n0FEB3E3VSHjyYMcFadtQORvl1Ssj&index=16

HTML in SignUp.html

```
{% extends "Home/base.html" %}
{#extends the base.html page. This shows the relationship with the parent page base.html and this child page#}
{% load static %}
{#load css file to use for styling#}
{% block css %}<link rel="stylesheet" type="text/css" href="{% static 'css/SignUp.css' %}">{% endblock %}
{#load css file as a link that can be read as a text or css file with reference to file location#}
{#block css used to link to block element in parent page to this page#}
{% block body %}
{#start of body elements#}
    <div class="Container">
        {# create div class called container #}
        <h1 align = "center">{% block title %}Signup{% endblock %}</h1>
        {#open tag for heading to be aligned in center of page. opening block title used to show that#}
        {#variable between block and endblock is used for title in parent page. endblock shows there is no more#}
        {#content for the given variable. End tag for heading to show end of heading#}
        <div class="Content" align= "center">
            {# create div with id of content and align the div to center of page #}
            <form method = "post">
                {#opening form tag to get post from page#}
                {% csrf_token %}
                {#this is a csrf_token which securely collects the information submitted in the form#}
                <table border="0">
                  {#start of table with invisible border#}
                    <tr><p>
                        {% for field in form%}
                            <br>{{field.label_tag}}
                            <br>{{ field }}
                        {% endfor %}
                        </p>
                        {%for field in form%}
                            {{field.errors}}
                        {% endfor %}
                        {{ form.non_field_errors }}
                        {#this displays all of the errors that will display in the form#}
                        <td>
                          {#start of column#}
                        <button type="submit" class=SignUpButton>Sign Up</button>
                        {#this is the signupbutton#}</td>
                    </tr>
```

```
                    {#end of row#}
                </table>
                {#end of table#}
            </form>
            {#end of form#}
        </div>
        {#end of div#}
    </div>
    {#end of div#}
{% endblock %}
{#end of body block#}
```

This displays the signup form. If the form is invalid when clicking the sign up button errors are displayed in the form. If the form is valid, clicking the sign up button will redirect users to the login page.

CSS in SignUp.css

```
h1{
    color:white;
    /*this makes the heading white*/
    font-size:3.25vw
    /*this makes the heading 3.25% of the size of the viewport width*/
}
.Container{
    border: 1px solid black;
    /*this makes the Containers border 1 pixel and black*/
    background-color: darkblue;
    /*this makes the Container have a darkblue background*/
    margin: auto;
    /*this makes the margin of the container automatic*/
    width: 640px
    /*this makes the width of the container 640px*/
}
.SignUpButton{
    display:inline-block;
    /*this makes the signup button able to have height and width values*/
    padding:auto;
    /*this makes the padding around the button automatically adjust*/
    font-size:3.5vw;
    /*this makes the font size of the button 3.5% of the viewport width*/
    cursor:pointer;
    /*this changes the cursor to a pointer when you hover over the button*/
    text-align: center;
    /*this makes the text in the button align to the center*/
    outline:black;
    /*this gives the text a black border*/
    color: white;
    /*this makes the text white*/
    background-color: red;
    /*this makes the button background red*/
    border:black;
    /*this makes the button have a black border*/
    border-radius: 3px;
    /*this gives the button a rounded effect of 3 pixels*/
}
```

```
.SignUpButton:hover{
    background-color: rgb(220,20, 60)
    /*when hovering over button gives the button this color*/
}
.SignUpButton:active{
    background-color: rgb(220,20, 60);
    /*when clicking button gives button this colour*/
    transform:translateY(2px)
    /*moves button 2 pixels down when clicked*/
}
@media screen and (max-width: 640px) {
    /*when page width is less than or equal to 640 pixels*/
    .Container{
        width: 100%;
        /*make width of container width of page*/
    }
}
@media screen and (max-height: 480px) {
    /*when page height is less than or equal to 480 pixels*/
    .Container{
        margin: auto;
        /*make margin of container automatically resize*/
    }
}
```

In Home/views.py

```
def SignUp(request):
    # function labeled SignUp which takes parameter request
    if request.method == 'POST':
        # if the request method is post
        form = RegisterForm(request.POST)
        # the registration form is called with parameter of request.post
        # this will load the registration form with the data filled in the form
        if form.is_valid():
            # if the form is valid
            form.save()
            # save the form
            return HttpResponseRedirect('Login')
            # return to the login page
    else:
        # if the form isnt valid
        form = RegisterForm()
        # display another blank registration form
    context = {
        'form': form,
    }
    # the context that will be displayed on the page is form
    return render(request, 'Home/SignUp.html', context)
    # returns by rendering the request as the SignUp.html with the relevant form
```

This imports the RegisterForm from Home/forms.py to allow users to register on the network. If the form is valid, the form is saved and the user is redirected to the login template. If the form is invalid, an empty form is displayed with validation errors. Render is used to render the page with the request and form via the render method imported in Home/views.py

SQL

INSERT INTO "Home_userinformation" ("user_id", "year_id", "interest_id", "postcode_id", "town_id") VALUES (%d, NULL, NULL, NULL, NULL)

In Home/forms.py[57] [58] [59] [60]

```
class RegisterForm(UserCreationForm):
    # creates registration form
    email = forms.EmailField(widget=forms.EmailInput(attrs={'placeholder': 'Email'}))
    # creates email field with placeholder of email
    first_name = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'Firstname'}))
    # creates first_name field as a character field with placeholder of Firstname
    last_name = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'Lastname'}))
    # creates last_namefield as a character field with placeholder Lastname
    username = forms.CharField(widget=forms.TextInput(attrs={'placeholder': 'Username'}))
    # creates username as character field with placeholder as username

    class Meta():
        model = User
        # connects to User table in database
        fields = ['email', 'username', 'password1', 'password2', 'first_name', 'last_name']
        # calls these fields to use in forms. Im not customising the password inputs so calling the fields is sufficient
    def clean(self):
        # creates method to clean data
        cleaned_data = super().clean()
        # stores cleaned data as a variable
        email = cleaned_data.get("email")
        first_name = cleaned_data.get("first_name")
        last_name = cleaned_data.get("last_name")
        username = cleaned_data.get("username")
        # gets cleaned data in form to use for validation
        with open("Home/banned words.txt") as file:
            # opens banned words text file as file
            lines = [line.strip() for line in file]
            # appends banned words to a list
        for word in lines:
            # for word in lines list
            if (word in email):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )
```

```
            elif (word in first_name):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )
            elif (word in last_name):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )
            elif (word in username):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )
                # raise a validation error if the word is any of the recieved inputs
        if User.objects.filter(email=email).exists():
            raise forms.ValidationError("That email is already assigned to a user")
            # raise a validation error is the email is already assigned to a user
        elif "@hfed.net" not in email:
            raise forms.ValidationError("This web app only allows hfed users")
            # raise a validation error if the user doenst have @hfed.net in their eail
        elif (any(char.isdigit() for char in first_name)) == True:
            raise forms.ValidationError("Remove the number from your first name")
            # raise a validation error if the firstname has a number in it
        elif (any(char.isdigit() for char in last_name)) == True:
            raise forms.ValidationError("Remove the number from your last name")
            # raise a validation error if the last name has a number in it
        return cleaned_data
        # return the cleaned data
```

This will display the fields in the form and check for any validation errors in the form. This uses the UserCreationForm imported in Home/forms.py and the User table imported in Home/forms.py to

[57] https://stackoverflow.com/questions/19859282/check-if-a-string-contains-a-number
[58] https://www.freewebheaders.com/full-list-of-bad-words-banned-by-google/
[59] https://stackoverflow.com/questions/3925614/how-do-you-read-a-file-into-a-list-in-python/3925701
[60] https://docs.djangoproject.com/en/2.1/ref/forms/validation/

create new users in the database. This inherits the attributes and methods of the model specified in class Meta and the fields specified are the fields that will be used in the form.

If a user is created the below code in Home/models.py is run:[61]

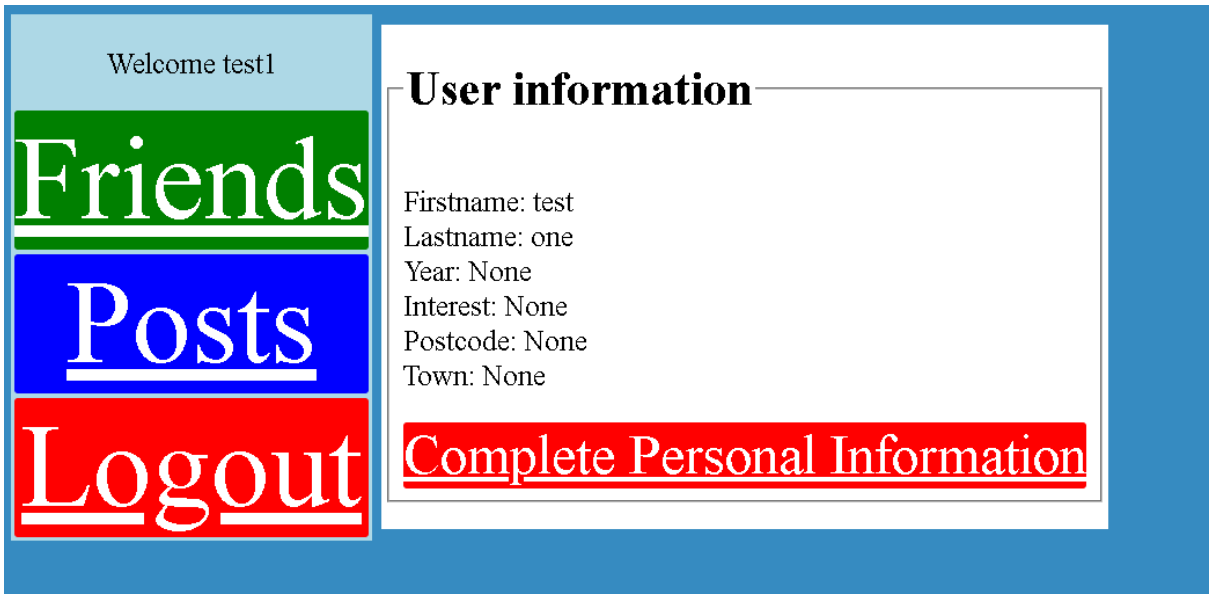
```
@receiver(post_save, sender=User)
# when a user signs up to social network
def create_user_information(sender, instance, created, **kwargs):
    # takes parameters to create a user profile
    if created:
        # if a user is created
        UserInformation.objects.create(user=instance)
        # will create a user profile associated to that user
```

In Home/urls.py

```
path('SignUp', views.SignUp, name='SignUp'),
```

This links the signup URL to the SignUp.html page

Profile



This page allows users to choose to complete their personal information and to get to the friends, posts and logout pages.

[61] https://github.com/maxg203/Django-Tutorials/blob/master/accounts/models.py

HTML in Profile.html

```
{% extends 'Home/base2.html'%}
{#extends the base2.html page. This shows the relationship with the parent page base2.html and this child page#}
{% load static %}
{#django funciton to load css styles for page#}
{% block title %}Profile{% endblock %}
{#title tag to store title of page#}
{% block css %}<link rel="stylesheet" type="text/css" href="{% static 'css/Profile.css' %}">{% endblock %}
{#link to the css file to load the style for this page#}
{% block body %}
{#blocks for body to link to body elements in parent page.#}
{% if user.is_authenticated %}
{#if the user is authenticated#}
<div class="container">
  <table>
    <tbody>
      <tr>
        <td>
      <table bgcolor = lightblue>
        {#table with a light blue background is displayed#}
        <tbody>
        <tr>
          <td>
            <p align = "center"> Welcome {{request.user.username}} </p>
            {#display welcome with username of user#}
            <a href="{% url 'Friends' %}" class="Button1">Friends</button>
              {#display button that links to friends page#}
          </td>
        </tr>
        <tr>
          <td>
            <a href="{% url 'Posts' %}" class="Button2">Posts</button>
            {#displays button that links to posts page#}
          </td>
        </tr>
        <tr>
          <td>
            <a href="{% url 'Logout' %}" class="Button3">Logout</button>
              {#displays button that links to logout page#}
          </td>
```

```
      </tr>
      </tbody>
    </table>
    </td>
      <td>
    </td>
      <td>
    <table bgcolor = white>
      {#in a table with a white backgorund#}
      <tbody>
      <tr>
        <td>
        {%block content%}
        <fieldset>
        <legend><h1>User information</h1></legend>
        {#display the user information of the user#}
          <p>
          Firstname: {{user.first_name}}<br>
          Lastname: {{user.last_name}}<br>
          Year: {{user.userinformation.year}}<br>
          Interest: {{user.userinformation.interest}}<br>
          Postcode: {{user.userinformation.postcode}}<br>
          Town: {{user.userinformation.town}}
          {#get users firstname, lastname, year, interest, postcode and town#}
          </p>
          {%if request.user.pk == user.pk%}
          {#if the user viewing the profile is the owner of the profile#}
            <a href="{% url 'EditProfile' %}" class="Button4">Complete Personal Information</button>
              {#display an edit profile button linking to edit the users profile#}
          {%else%}
            <a href="{% url 'AddFriend' %}" class="Button4">Go Back</button>
              {#return to the AddFriend page as you can only view other profile pages from AddFriend#}
          {%endif%}
        </fieldset>
        {%endblock%}
        </td>
      </tr>
      </tbody>
    </table>
```

```
      </td>
      </tr>
    </tbody>
  </table>
  {% else %}
  <p>You can't access this page as you aren't logged in</p>
  {#display this error message is the user viewing the page is not logged in#}
  {% endif %}
</div>
{% endblock %}
```

CSS in Profile.css

```
.Button4{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:3.5vw;
    /*make font size of button 3.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: red;
    /*have background of button set to red*/
    border:black;
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
.Button4:hover{
    background-color: rgb(220,20, 60)
    /*have button set to lightred when hovering over button*/
}
.Button4:active{
    background-color: rgb(220,20, 60);
    /*when button clicked changed color to lightred*/
    transform:translateY(2px)
    /*move button down 2 pixels*/
}
legend{
    font-size:1.5vw
    /*makes legend have a font size of 1.5% of the viewport width*/
}
```

In Home/views.py[62]

```
def Profile(request, pk=None):
    # function labeled Profile which takes parameters request and pk which is None
    if pk:
        # if a pk is supplied
        user = User.objects.get(pk=pk)
        # get the users data for their profile
    else:
        user = request.user
        # display the logged users profile
    context = {'user': user}
    # take user as a parameter in context
    return render(request, 'Home/Profile.html', context)
    # returns by rendering the request as the Profile.html with the relevant user data
```

This will either display the currently logged in users profile or the profile of other users on the social network. User is the User table in the database. Render is used to render the page with the request and users data via the render method imported in Home/views.py.

SQL

SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user" WHERE "auth_user"."id" = %d

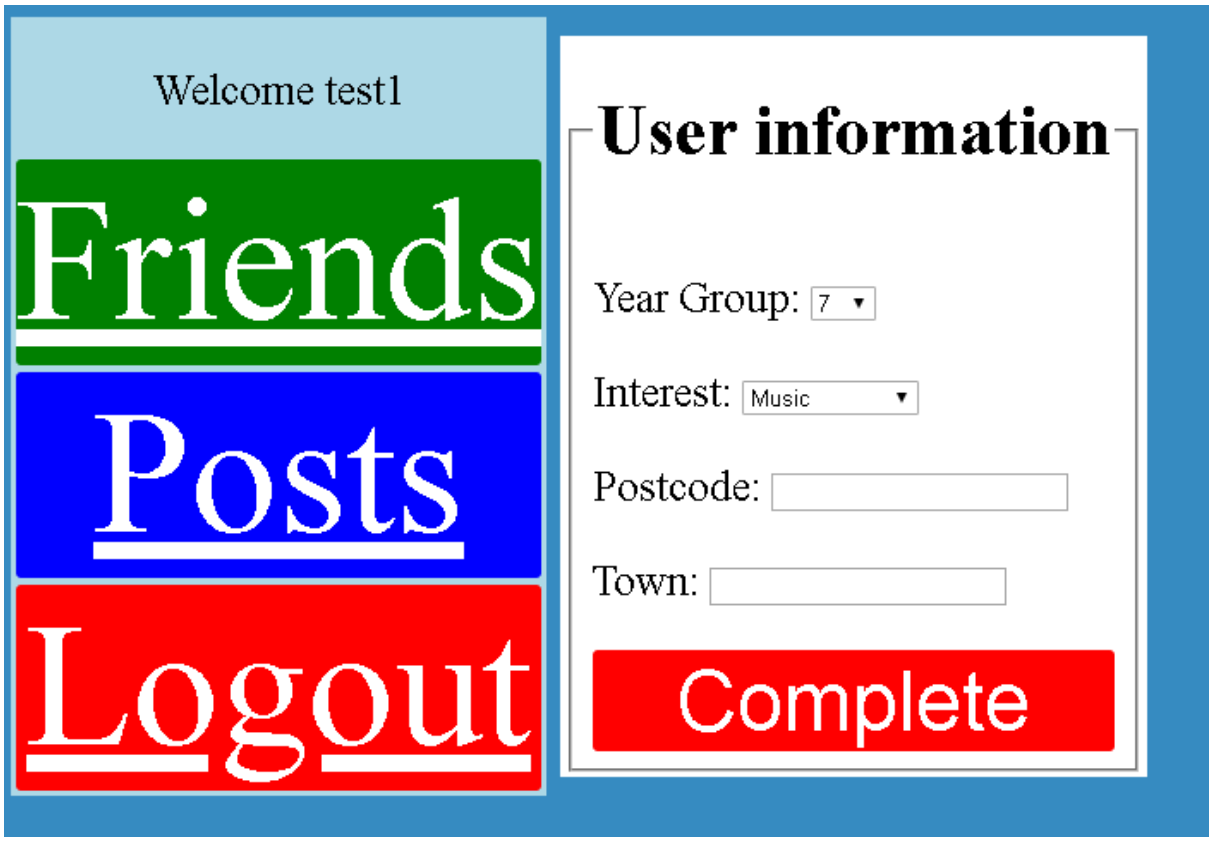
In Home/urls.py

```
path('Profile', views.Profile, name='Profile'),
path('Profile/<int:pk>', views.Profile, name='ViewOtherProfile'),
```

The first URL connects the user to their own profile. The second URL connects the user to another users' profile.

[62] https://www.youtube.com/watch?v=2yoWf-kDXIk&list=PLw02n0FEB3E3VSHjyYMcFadtQORvl1Ssj&index=35

Complete Profile Page



This page allows users to fill out their user information and to either get to the friends page, posts page or to logout of the social network.

HTML in Edit Profile.html

```
{%extends 'Home/Profile.html'%}
{#extends the Profile.html page. This shows the relationship with the parent page Profile.html and this child page#}
{%block content%}
<form method="POST">
    {% csrf_token %}
    {#this is a csrf_token which securely collects the information submitted in the form#}
    <fieldset>
    <legend><h1>User information</h1></legend>
        {{form1.as_p}}
        {{form2.as_p}}
        {{form3.as_p}}
        {{form4.as_p}}
        {#displays forms in paragraph format#}
        <button type="submit" class="Button4">Complete</button>
        {#complete button to submit form#}
    </fieldset>
</form>
{%endblock%}
```

In Home/views.py

```
def EditProfile(request):
    # function labeled EditProfile which takes parameter request
    if request.method == 'POST':
        # if form is displayed
        form1 = YearGroupForm(request.POST, instance=request.user.userinformation.year)
        form2 = InterestsForm(request.POST, instance=request.user.userinformation.interest)
        form3 = PostCodeForm(request.POST, instance=request.user.userinformation.postcode)
        form4 = TownForm(request.POST, instance=request.user.userinformation.town)
        # initialise values for each form as an instance in relation to the user
        if form1.is_valid() and form2.is_valid() and form3.is_valid() and form4.is_valid():
            # if all the forms are valid
            data1 = request.user  # get the current users data
            data2 = form1.cleaned_data['year']
            data3 = form2.cleaned_data['interest']
            data4 = form3.cleaned_data['postcode']
            data5 = form4.cleaned_data['town']
            # store the cleaned data returned from the forms
            year = YearGroup.objects.get(year=data2)
            interest = Interests.objects.get(interest=data3)
            postcode = PostCode.objects.get(postcode=data4.upper())
            town = Town.objects.get(town=data5.lower())
            # get the primary key of the object in their database tables
            user = UserInformation.objects.get(user=data1)
            # get the users object from the database table for UserInformation
            user.year = year
            user.interest = interest
            user.postcode = postcode
            user.town = town
            user.save()
            # save the primary keys of objects returned as the foreign keys in the UserInformation table
            # for that user
            return HttpResponseRedirect('../Profile')
            # redirect the user back to the profile page
    else:
        form1 = YearGroupForm(instance=request.user.userinformation.year)
        form2 = InterestsForm(instance=request.user.userinformation.interest)
        form3 = PostCodeForm(instance=request.user.userinformation.postcode)
        form4 = TownForm(instance=request.user.userinformation.town)
        # if there was an error in the form display errors and a blank form
    context = {
        'form1': form1,
        'form2': form2,
        'form3': form3,
        'form4': form4
    }  # store data to be used in the html template
    return render(request, 'Home/Edit Profile.html', context)
    # returns by rendering the request as the Edit Profile.html with the relevant forms
```

This will display the edit profile page for users to enter their year group, interests, postcode and town. This will then store the details in their relevant database tables and will associate the object in the database to the user. If the form is not valid then validation errors are raised. YearGroupForm, InterestsForm, TownForm and PostCodeForm are the forms which are imported from Home/forms.py. YearGroup, Interests, PostCode, Town and UserInformation are database tables imported from Home/models.py. HttpResponseRedirect is imported in Home/views.py to display a URL page after a form is submitted. Render is imported in Home/views.py to display a page associated with the function.

SQL to insert record into YearGroup

INSERT INTO "Home_yeargroup" ("year") VALUES (%d)

SQL to insert record into Interests

INSERT INTO "Home_interests" ("interest") VALUES (\'%s\')

SQL to insert record into Postcode

INSERT INTO "Home_postcode" ("postcode") VALUES (\'%s\')

SQL to insert record into Town

INSERT INTO "Home_town" ("town") VALUES (\'%s\')

SQL to select record from YearGroup

SELECT "Home_yeargroup"."id", "Home_yeargroup"."year" FROM "Home_yeargroup" WHERE "Home_yeargroup"."year" = %d'

SQL to select record from Interest

SELECT "Home_interests"."id", "Home_interests"."interest" FROM "Home_interests" WHERE "Home_interests"."interest" = \'%s\'

SQL to select record from Postcode

SELECT "Home_postcode"."id", "Home_postcode"."postcode" FROM "Home_postcode" WHERE "Home_postcode"."postcode" = \'%s\'

SQL to select record from Town

SELECT "Home_town"."id", "Home_town"."town" FROM "Home_town" WHERE "Home_town"."town" = \'%s\'

SQL to select user from User and UserInformation

SELECT "Home_userinformation"."id", "Home_userinformation"."user_id", "Home_userinformation"."year_id", "Home_userinformation"."interest_id", "Home_userinformation"."postcode_id", "Home_userinformation"."town_id" FROM "Home_userinformation" WHERE "Home_userinformation"."user_id" = %d'

SQL to update UserInformation for user

UPDATE "Home_userinformation" SET "user_id" = %d, "year_id" = %d, "interest_id" = %d, "postcode_id" = %d, "town_id" = %d WHERE "Home_userinformation"."id" = %d'

In forms.py

```
class YearGroupForm(forms.ModelForm):
    # creates year group form
    options = [(year, year) for year in range(7, 14)]
    # stores options to select for yeargroup
    year = forms.ChoiceField(label="Year Group", choices=options)
    # creates dropdown field with label for yeargroup with the specified choices

    class Meta():
        model = YearGroup
        # connects to YearGroup table in database
        fields = ('year',)
        # accessed field is year

    def clean(self):
        # defines clean method to clean data
        cleaned_data = super().clean()
        # stores cleaned data as a variable
        year = cleaned_data.get("year")
        # gets cleaned data from form
        YearGroup.objects.get_or_create(year=year)
        # gets or creates the object in the YearGroup table
        return cleaned_data
        # returns the cleaned data
```

This displays that dropdown field for year group to be selected between 7 and 13. This will also create the instance of the year group in the database table for YearGroup if the instance does not exist. This inherits the attributes and methods of the model specified in class Meta and the fields specified are the fields that will be used in the form.

```
class InterestsForm(forms.ModelForm):
    # creates interests form
    choice = (("Music", "Music"), ("Reading", "Reading"), ("Tv or Movies", "Tv or Movies"), ("Video Games", "Video Games"), ("Art", "Art"),)
    # stores choices for interests
    interest = forms.ChoiceField(choices=choice, label="Interest")
    # creates dropdown for interests with label as interest

    class Meta():
        model = Interests
        # connects to interests table in database
        fields = ("interest",)
        # accessed field is interest

    def clean(self):
        # creates clean method to clean data
        cleaned_data = super().clean()
        # stores cleaned data as a varaiable
        interest = cleaned_data.get("interest")
        # get cleaned data from form
        Interests.objects.get_or_create(interest=interest)
        # get or create object in interest table
        return cleaned_data
        # returns cleaned data
```

This displays the dropdown field for interests to be selected. This will also create the instance of the interest in the Interests table if the instance does not exist. This inherits the attributes and methods of the model specified in class Meta and the fields specified are the fields that will be used in the form.

```python
class PostCodeForm(forms.ModelForm):
    # creates postcode form
    postcode = forms.CharField(max_length=7, min_length=6, label="Postcode")
    # postcode is a character field with length between 6 and 7 characters and has the label postcode

    class Meta():
        model = PostCode
        # connects to the PostCode table in the database
        fields = ('postcode',)
        # accessed field is postcode

    def clean(self):
        # creates clean method to clean data
        cleaned_data = super().clean()
        # store cleaned data as variable
        postcode = cleaned_data.get("postcode")
        # get cleaned data from form
        with open("Home/banned words.txt") as file:
            # open banned words as a text file
            lines = [line.strip() for line in file]
            # store words in text file as a list
        for word in lines:
            # for each word in the banned words list
            if (word in postcode):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )  # raise a validation error is the word is in the field
        PostCode.objects.get_or_create(postcode=postcode.upper())
        # get or create object in PostCode table
        return cleaned_data
        # return cleaned data
```

This will display the input field for Postcode which will ensure the postcode is between 6 and 7 characters and will ensure that validation errors are raised if the postcode has banned words. This will also create an instance of the postcode in the PostCode table if it does not exist. This inherits the attributes and methods of the model specified in class Meta and the fields specified are the fields that will be used in the form.

```
class TownForm(forms.ModelForm):
    # creates town form
    town = forms.CharField(label="Town")
    # town is a character field with label town

    class Meta():
        model = Town
        # connects to Town table in database
        fields = ('town',)
        # accessed field is Town

    def clean(self):
        # creates clean method to clean data
        cleaned_data = super().clean()
        # stores cleaned data in a variable
        town = cleaned_data.get("town")
        # get cleaned data from form
        with open("Home/banned words.txt") as file:
            # open banned words as a text file
            lines = [line.strip() for line in file]
            # store banned words in a list
        for word in lines:
            # for each word in the list
            if (word in town):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )  # raise a validation error if a banned word is in the field
        Town.objects.get_or_create(town=town.lower())
        # get or create an object in the Town table
        return cleaned_data
        # return cleaned data
```

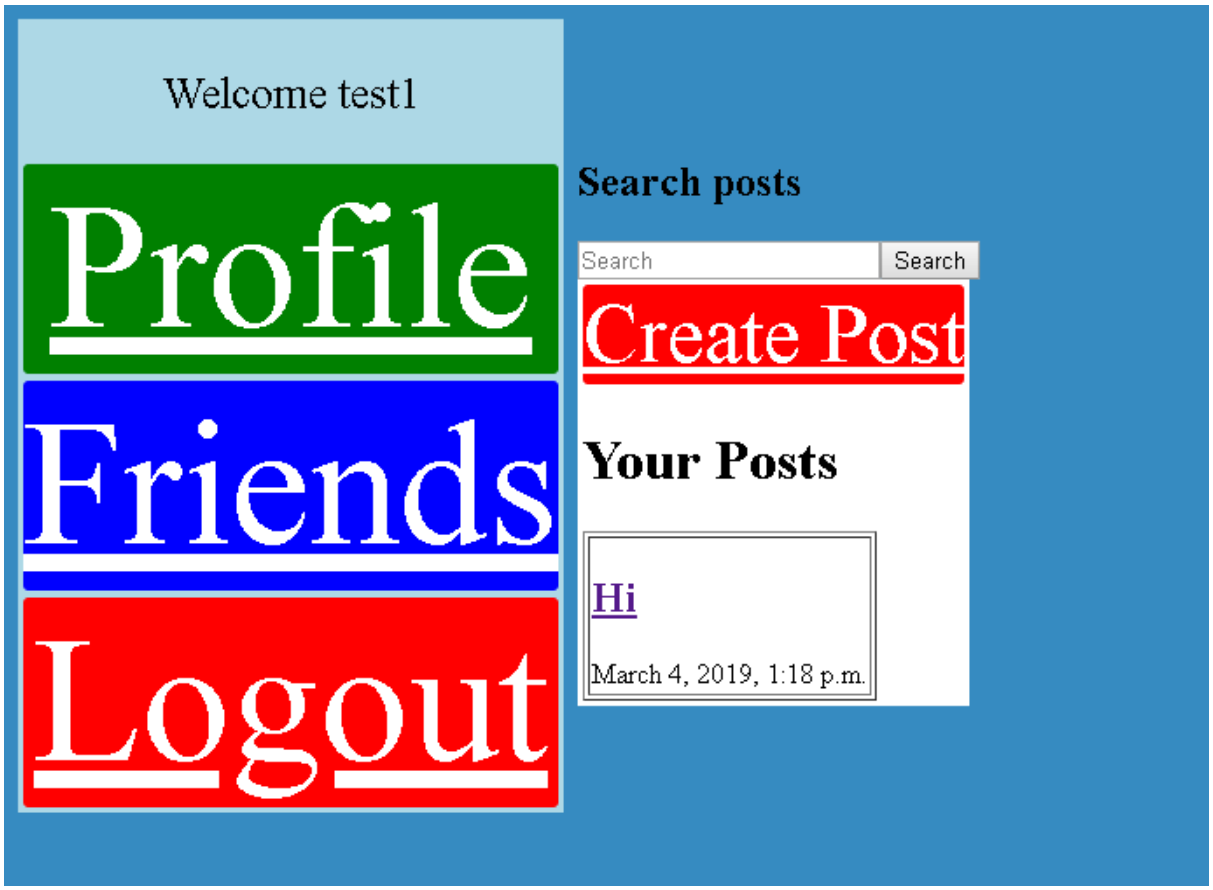
This will create the input field for town which will raise a validation error if the field has any banned words in it. Additionally, this will also create an instance of the town in the Town table if it does not exist. This inherits the attributes and methods of the model specified in class Meta and the fields specified are the fields that will be used in the form.

In Home/urls.py

```
path('Profile/Edit', views.EditProfile, name='EditProfile'),
```

This assigns the URL Profile/Edit to the Edit Profile.html page

Posts



This page allows users to search for posts, create posts, view their own posts or other users posts and allows users to view the profile page, friends page and to logout.

HTML in Posts.html

```
{% extends 'Home/base2.html'%}
{#extends the base.html page. This shows the relationship with the parent page base2.html and this child page#}
{% load static %}
{#django funciton to load css styles for page#}
{% block title %}Posts{% endblock %}
{#title tag to store title of page#}
{% block css %}<link rel="stylesheet" type="text/css" href="{% static 'css/Posts.css' %}">{% endblock %}
{#link to the css file to load the style for this page#}
{% block body %}
{#blocks for body to link to body elements in parent page.#}
{% if user.is_authenticated %}
{#if the user is authenticated#}
<div class="container">
  <table>
    <tbody>
      <tr>
        <td>
      <table bgcolor = lightblue>
        {#table with a light blue background is displayed#}
        <tbody>
        <tr>
          <td>
            <p align = "center"> Welcome {{user.username}} </p>
            {#display welcome with username of user#}
            <a href="{% url 'Profile' %}" class="Button1">Profile</button>
              {#display button that links to profile page#}
          </td>
        </tr>
        <tr>
          <td>
            <a href="{% url 'Friends' %}" class="Button2">Friends</button>
            {#display button that links to friends page#}
          </td>
        </tr>
        <tr>
          <td>
            <a href="{% url 'Logout' %}" class="Button3">Logout</button>
              {#displays button that links to logout page#}
          </td>
```

```
  </tr>
  </tbody>
</table>
</td>
  <td>
</td>
  <td>
{%block content%}
<table bgcolor = white>
  {#in a table with a white backgorund#}
  <tbody>
  <tr>
    <td>
     <a href="{% url 'CreatePost' %}" class="Button4">Create Post</button>
       {#display an create post button linking to create a post#}
    </td>
  </tr>
  <h2>Search posts</h2>
  {#search posts in heading tags#}
  <tr>
    <form method="GET" action="{% url 'Search' %}">
      <input name="query" value="{{request.GET.query}}" placeholder="Search">
      <button type="submit">Search</button>
    </form>
    {#form used to search posts which links to search#}
    {%if post%}
    {#if you have made any posts will display content below#}
    <td>
      <h1>Your Posts</h1>
      {#Your posts in heading tags#}
      {% for post in post %}
      {#for every post you have posted#}
        <table border = 1>
          <tr>
            <td>
            <h2><a href="{% url 'PostDetail' pk=post.pk %}">{{post.title}}</a></h2>{{post.publish}}<br>
            {#display the post with a link to view the post on a seperate page#}
            {#displays posts title and date and time post was published#}
```

```
                {% if post.image %}
                {#if the post has an image associated to it#}
                <img src="{{post.image.url}}" width="240"/>
                {#will display the posts image with a url to the image#}
                {%endif%}
              </td>
            </tr>
          </table>
        {% endfor %}
      </td>
      {%endif%}
      {%if others%}
      {#if other users have made a post#}
      <td>
        <h1>Other Posts</h1>
        {#Other posts in heading tags#}
        {% for post in others%}
        {#for every post another user has made#}
          <table border = 1>
            <tr>
              <td>
              <h2><a href="{% url 'PostDetail' pk=post.pk %}">{{post.title}}</a></h2>{{post.publish}}<br>
              {#post is displayed with title and date and time it was published#}
              {#can view post on a seperate page#}
                {% if post.image %}
                {#if the post has an image associated to it#}
                <img src="{{post.image.url}}" width="240"/>
                {#displays posts image with a url to the image#}
                {%endif%}
              </td>
            </tr>
          </table>
        {% endfor %}
      </td>
      {%endif%}
    </tr>
    </tbody>
  </table>
  {%endblock%}
```

```
      </td>
      </tr>
    </tbody>
  </table>
  {% else %}
  <p>You can't access this page as you aren't logged in</p>
  {#display this error message is the user viewing the page is not logged in#}
  {% endif %}
</div>
{% endblock %}
```

CSS in Posts.css

```
.Button4{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:3.5vw;
    /*make font size of button 3.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: red;
    /*have background of button set to red*/
    border:black;
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
.Button4:hover{
    background-color: rgb(220,20, 60)
    /*have button set to lightred when hovering over button*/
}
.Button4:active{
    background-color: rgb(220,20, 60);
    /*when button clicked changed color to lightred*/
    transform:translateY(2px)
    /*move button down 2 pixels*/
}
.Button5{
    display:inline-block;
    /*allow height and width values to be made for button*/
    width: 100%;
    font-size:3.5vw;
    /*make font size of button 3.5% of viewport width*/
    cursor:pointer;
    /*have cursor change to pointer on button hover*/
```

```
    text-align: center;
    /*align button text to center of button*/
    outline:black;
    /*set outline of text to black*/
    color: white;
    /*have text color set to white*/
    background-color: blue;
    /*have background of button set to red*/
    border:black;
    /*have corder of button set to black*/
    border-radius: 3px;
    /*have border of buton set to 3 pixels and rounded*/
}
.Button5:hover{
    background-color: lightblue
    /*have button set to lightred when hovering over button*/
}
.Button5:active{
    background-color: lightblue;
    /*when button clicked changed color to lightred*/
    transform:translateY(2px)
    /*move button down 2 pixels*/
}
```

In Home/views.py

```
def Posts(request):
    # creates function labeled Posts with parameter request
    post = Post_model.objects.filter(user=request.user).order_by('-publish')
    # orders the users posts by most recently created
    others = Post_model.objects.all().exclude(user=request.user).order_by('-publish')
    # orders other users posts by most recently created
    content = {
        'post': post,
        'others': others
    }  # store users posts as variables to be used in the template
    return render(request, 'Home/Posts.html', content)
    # render Posts.html with the associated users posts
```

This gets all the current users posts in the database and orders them by the most recently created post and gets all the other users posts in the database and orders them by the most recently created post. Post_model refers to the Post table in Home/models.py imported in Home/views.py. Render is imported in Home/views.py to display a page associated with the function.

SQL

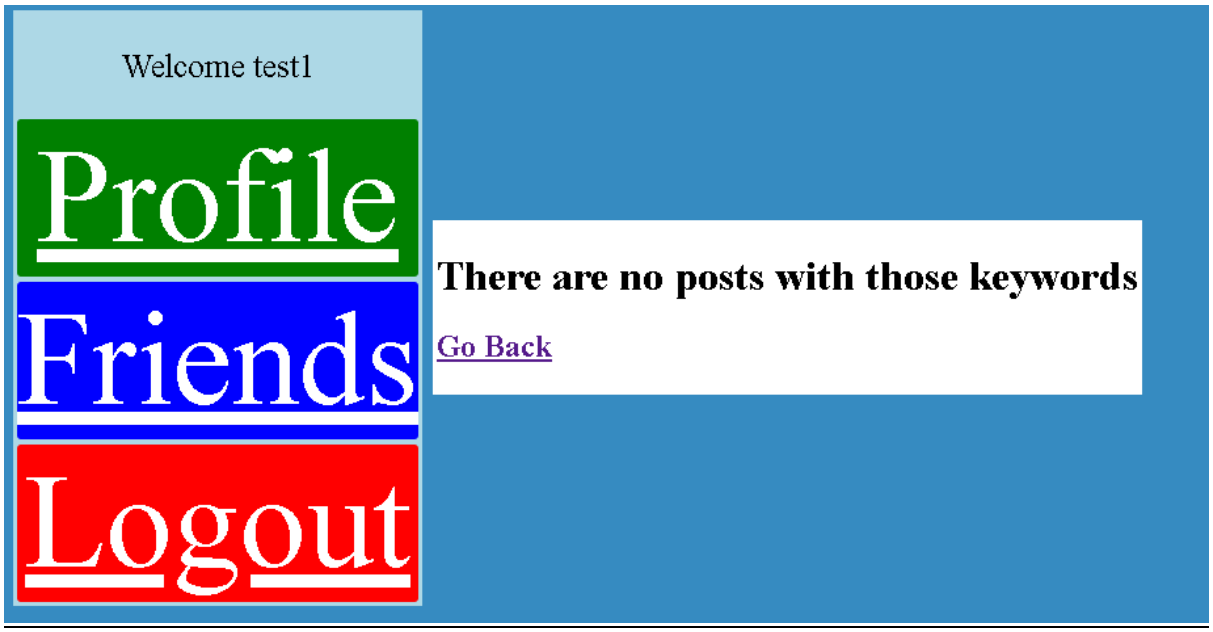
SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user" WHERE "auth_user"."id" = %d'

In Home/urls.py

```
path('Posts', views.Posts, name='Posts'),
```

This links the URL Posts to the Posts.html file.

If a user searches for a post:



This page is displayed if a user searches for a post that does not exist or will display posts that exist in the database. From this page a user will be able to go back to view all the posts they have created recently or users can view the profile page, friends' page or logout.

HTML in search.html

```
{%extends 'Home/Posts.html'%}
{#extends the Posts.html page. This shows the relationship with the parent page Posts.html and this child page#}
{%block content%}
  <table bgcolor = white>
    {#in a table with a white backgorund#}
      <tbody>
        <td>
          {%if results%}
          {#if there are results#}
              {% for post in results %}
              {#for every post in the results#}
                <table border = 1>
                  <tr>
                    <td>
                    <h2><a href="{% url 'PostDetail' pk=post.pk %}">{{post.title}}</a></h2>{{post.publish}}<br>
                    {#display the post with its title and date it was published with a link to the posts page#}
                      {% if post.image %}
                      {#if the post has an image#}
                      <img src="{{post.image.url}}" width="240"/>
                      {#display the posts image#}
                      {%endif%}
                    </td>
                  </tr>
                </table>
              {% endfor %}
          {%else%}
            <h1>There are no posts with those keywords</h1>
            {#if there are no results display this error message#}
          {%endif%}
          <a class="button" href = "{% url 'Posts' %}"><h2>Go Back</h2></a>
        </td>
      </tbody>
  </table>
{%endblock%}
```

In Home/views.py[63]

```
def Search(request):
    # create search function with parameter request
    query = request.GET.get('query')
    # store search parameters as a query
    if query:
        # if a query has been made when searching
        results = Post_model.objects.filter(Q(title__icontains=query) | Q(content__icontains=query))
        # search for posts with the query in the title or contents of the posts
        # store the collected data as results
    else:
        results = Post_model.objects.all()
        # get all the posts in the database if no query has been made
    context = {
        "results": results,
    }  # store results as a parameter to be used in the template
    return render(request, "Home/search.html", context)
    # render the search template with all the collected results
```

This retrieves the query submitted in the search field on the Posts page and filters the objects in the Posts table to locate if there are any posts with that title or content otherwise the results displayed are just every post in the database. Post_model refers to the Posts table in the database located in Home/models.py. Q is a method imported in Home/views.py which will let me query the database with a query defined by a user in a form. Render is imported in Home/views.py to display a page associated with the function.

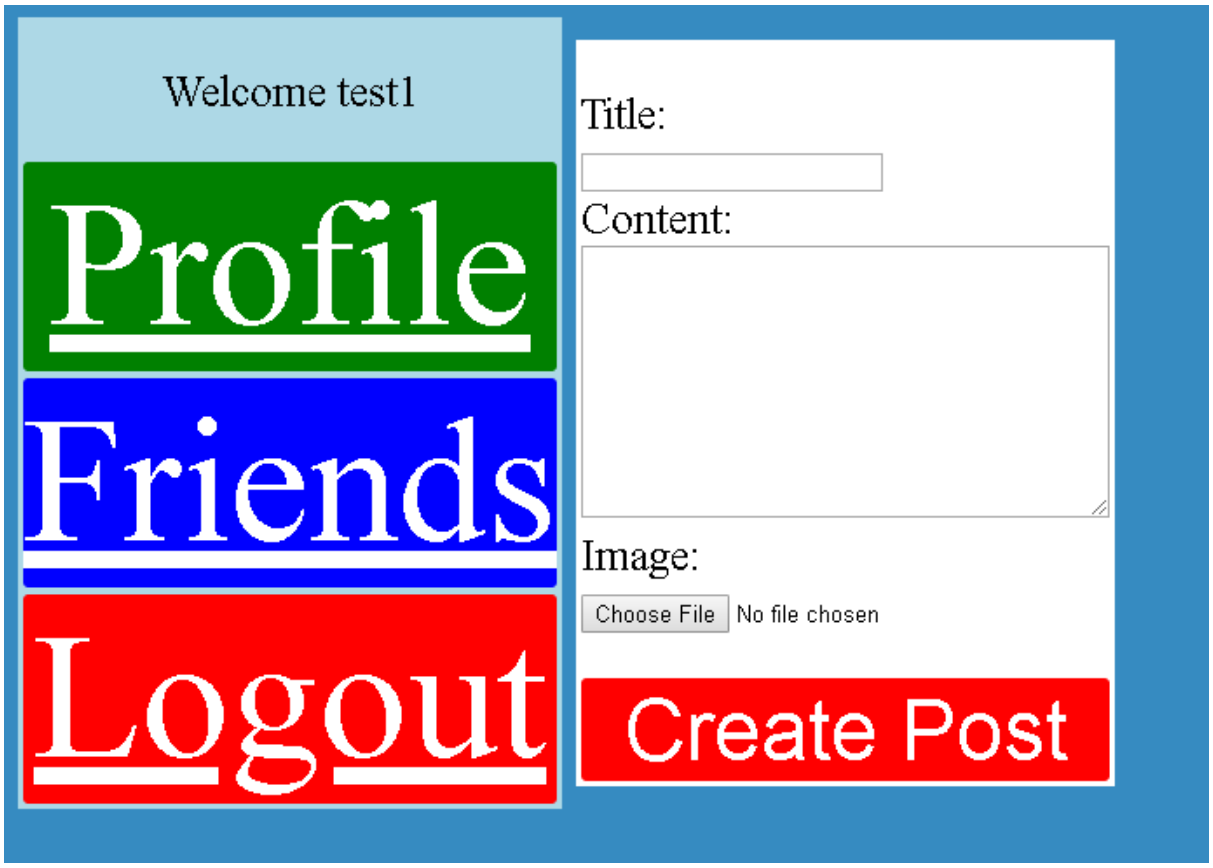
[63] https://www.youtube.com/watch?v=eyAIAZr5Q3w&index=13&list=WL&t=0s

In urls.py

```
path('Posts/results/', views.Search, name='Search'),
```

This links the URL Posts/results to the posts that will be displayed on the Search.html page.

Create Post



This allows users to create a post with an image by submitting the form or to view the profile and friends page or to logout.

HTML in Create Post.html

```
{%extends 'Home/Posts.html'%}
{#extends the Posts.html page. This shows the relationship with the parent page Posts.html and this child page#}
{%block content%}
<form method="POST" enctype='multipart/form-data'>
  {#this is a form used to recieve files#}
    {% csrf_token %}
    {#this is a csrf_token which securely collects the information submitted in the form#}
    <table bgcolor = white>
      {#in a table with a white backgorund#}
        <tbody>
        <tr>
            <td><p>
                {% for field in form %}
                    {{field.label_tag}}<br>
                    {{field}}<br>
                {% endfor %}
                </p>
                {%for field in form%}
                    {{field.errors}}
                {% endfor %}
                {{ form.non_field_errors }}
                {#display form and field errors#}
                <button type="submit" class="Button4">Create Post</button>
                {#submit button to submit post#}
            </td>
        </tr>
        </tbody>
    </table>
</form>
{%endblock%}
```

In Home/views.py

```
def CreatePost(request):
    # function labeled CreatePost taking parameter request
    if request.method == 'POST':
        # if the request has been made
        form = PostsForm(request.POST or None, request.FILES or None)
        # create an instance of the posts form to be viewed
        if form.is_valid():
            # if the form is valid
            post = form.save(commit=False)
            # get the post to edit a field in the Post table before its saved
            post.user = request.user
            # assign the user to the post
            post.save()
            # save the post
            return HttpResponseRedirect('../Posts')
            # redirect to the Posts page
    else:
        form = PostsForm()
        # if the post isnt valid show field errors and a blank form
    context = {
        'form': form
    }  # store form in context to be used in template
    return render(request, 'Home/Create Post.html', context)
    # render Create Post.html with the associated form
```

This gets the form to create a post from Home/forms.py under PostsForm. If the form is valid then the post will be saved in the database and assigned to the user which created the form but if the

form isn't valid, validation errors are raised. HttpResponseRedirect is imported in Home/views.py to display a URL page after a form is submitted. Render is imported in Home/views.py to display a page associated with the function.

SQL

INSERT INTO "Home_posts" ("user_id", "title", "content", "image", "publish") VALUES (%d, \'%s\', \'%s\', \'%s\', \'%d\')

In Home/forms.py

```
class PostsForm(forms.ModelForm):
    # creates posts form

    class Meta():
        model = Posts
        # connects to Posts table in database
        fields = ('title', 'content', 'image',)
        # accessed fields are title, content and image
        # title and content are text fields, image is an image field

    def clean(self):
        # creates method to clean data
        cleaned_data = super().clean()
        # stores cleaned data as a variable
        title = cleaned_data.get("title")
        content = cleaned_data.get("content")
        # gets cleaned data from form
        with open("Home/banned words.txt") as file:
            # open banned words are a text file
            lines = [line.strip() for line in file]
            # store each word in list
        for word in lines:
            # each each word in list
            if (word in title):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )
            elif (word in content):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )
            #raise validation error if banned word is in one of the fields
```

This displays the form with the title, content and image fields and will raise a validation error is any banned words are in the title and content fields. This links to the Posts table to get the field types for the input fields in the form. This inherits the attributes and methods of the model specified in class Meta and the fields specified are the fields that will be used in the form.

If an image is uploaded with the post then the below code is accessed:

- In GalaxyOnline/settings.py

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'Home/media')
# this stores the media files in a media directory
```

- In GalaxyOnline/urls.py

```
urlpatterns = [
    path('', include('Home.urls')),
    path('admin/', admin.site.urls),
    # this allows my apps urls and the admin urls to be used
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
# added static line from django docs and allows images to be stored in MEDIA_ROOT
```

- In Home/models.py under Posts model

```
image = models.ImageField(upload_to='post_images', blank=True, null=True)
# installed pillow to use image field
# image is a image field which will upload an image to post_images
# this field can be left blank
```

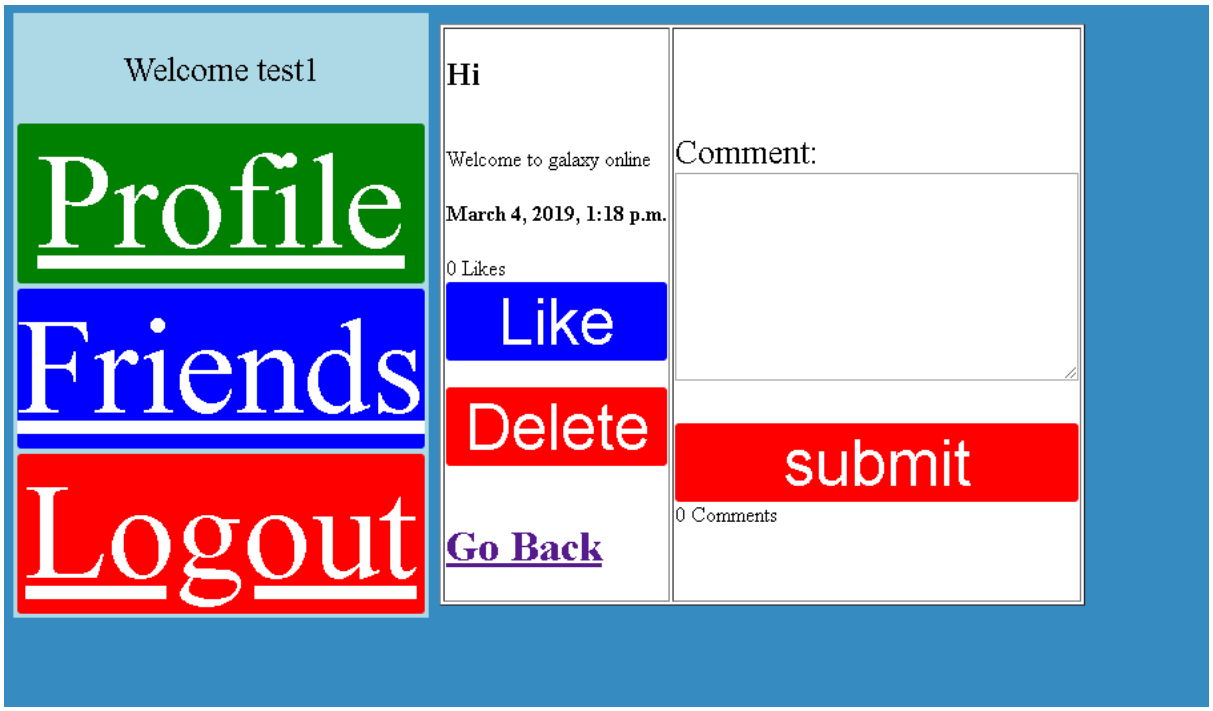
This allows images to be stored in Home/media/post_images when images are uploaded alongside posts.

In Home/urls.py

```
path('Create Post', views.CreatePost, name='CreatePost'),
```

This links the Create Posts URL to the Create Posts.html page.

Post detail[64]



This will allow users to view the posts made by other users and by themselves. Users will be able to like posts, comment on posts and delete their own posts or go back to the Posts page to see all the

64 https://www.youtube.com/watch?v=zg75OzTVOnU&list=PLEsfXFp6DpzQFqfCur9CJ4QnKQTVXUsRy&index=17

posts that are listed. Additionally users will be able to view the profile page, friends' page and logout.

HTML in post_detail.html

```
{%extends 'Home/Posts.html'%}
{#extends the Posts.html page. This shows the relationship with the parent page Posts.html and this child page#}
{%block content%}
<table bgcolor = white border=1>
  {#in a table with a white backgorund#}
        <tbody>
        <tr>
            <td>
            <h2>{{post.title}}</h2>
            {#display the posts title#}
            {{post.content}}<br>
            {#display the content in the post#}
            {% if post.image %}
                <img src="{{post.image.url}}" width="360"/><br>
            {%endif%}
            {#display the posts image if the post has an image#}
            {%if request.user != post.user%}
              <br>Made by {{post.user}}
            {%endif%}
            {#if the current user didnt create the post show who the post was made by#}
            <h4>{{post.publish}}</h4>
            {#display the time the post was published#}
            {{total_likes}} Like{{total_likes|pluralize}}
            {#display the number of likes on the post#}
            <form action="{% url 'LikePost'%}" method="POST">
              {#link to like post#}
                {% csrf_token %}
                {#this is a csrf_token which securely collects the information submitted in the form#}
                {%if is_liked%}
                    <button type="submit" name="post_id" value="{{post.id}}" class="Button4">Dislike</button>
                    {#if the user liked the post display a dislike button#}
                {%else%}
                    <button type="submit" name="post_id" value="{{post.id}}" class="Button5">Like</button>
                    {#if the user hasnt liked the post display a like button#}
                {%endif%}
            </form>
            {%if request.user == post.user%}
            {#if the current user made by post#}
                <form action="{% url 'DeletePost'%}" method="POST">
```

```
                {#link to delete post#}
                  {% csrf_token %}
                  {#this is a csrf_token which securely collects the information submitted in the form#}
                  <br>
                      <button type="submit" name="post_id" value="{{post.id}}" class="Button4">Delete</button>
                      {#button to delete post#}
              </form>
          {%endif%}
          <br>
          <a class="button" href = "{% url 'Posts' %}"><h1>Go Back</h1></a>
          {#button to return to posts page#}
          </td>
          <td>
              <form method="POST">
                  {% csrf_token %}
                  {#this is a csrf_token which securely collects the information submitted in the form#}
                  <p>
                  {% for field in form %}
                      Comment:<br>
                      {{field}}<br>
                  {% endfor %}
                  </p>
                  {%for field in form%}
                      {{field.errors}}
                  {% endfor %}
                  {{ form.non_field_errors }}
                  {#displays comment form#}
                  <button type="submit" class="Button4">submit</button>
                  {#button to submit comment#}
              </form>
              {{comments.count}} Comment{{comments|pluralize}}
              {#count number of comments for the post#}
              {% for comment in comments%}
              {#for each comment#}
                  <p class="mb-0">{{comment.content}}</p> {#display the comment#}
                  <footer class="blockquote-footer">by <cite title="Source Title">{{comment.user|capfirst}}</cite></footer>
                  {#display author of comment#}
              {%endfor%}
```

```
            </td>
        </tr>
        </tbody>
</table>
{%endblock%}
```

In Home/views.py

```
def PostDetail(request, pk):
    # create function labeled PostDetail with parameters request and pk
    post = get_object_or_404(Post_model, pk=pk)
    # get the object from the Post_model with that primary key and stpre it as post
    comments = Comment.objects.filter(post=post).order_by('-id')
    # get comments for post and order comments by id of user
    is_liked = False
    # store boolean
    comment_form = CommentForm()
    # get CommentForm
    if post.likes.filter(id=request.user.id).exists():
        # filter the posts likes field to see if user has liked post
        is_liked = True
        # update boolean
    elif request.method == 'POST':
        # if the request is post
        comment_form = CommentForm(request.POST or None)
        # get CommentForm to be displayed
        if comment_form.is_valid():
            # if the form is valid
            content = request.POST.get('content')
            # get the content in the form
            comment = Comment.objects.create(post=post, user=request.user, content=content)
            # create a object in the comments table assigniing the post, user and content in the object
            comment.save()
            # save the comment in the table
            return HttpResponseRedirect(post.get_absolute_url())
            # refresh the page with the new comment
    context = {
        'post': post,
        'is_liked': is_liked,
        'total_likes': post.total_likes(),
        'comments': comments,
        'form': comment_form,
    }  # stores post, is_likes, post.total_likes, comments and comment_form to be used in the template
    return render(request, 'Home/post_detail.html', context)
    # render the post_detail page with the contextual variables
```

This will display the specific post that a user has clicked on. Get_object_or_404 is a method imported in Home/views.py to check if an object in a database table exists, if the object does not exist then a 404 page is displayed. Post_model refers to the Posts table in the database. Comment refers to the comment table in the database, if a comment is made the comment is saved to the database while assigning the comment the post and user that the comment is linked to, the page is refreshed to show the new comment on the page. If the comment made is not valid, validation errors are raised and an empty comment form is displayed. Filtering the posts for likes checks if the user has liked the post thus choosing if the user should see a like or dislike button on the page. HttpResponseRedirect is imported in Home/views.py to display a URL page after a form is submitted. Render is imported in Home/views.py to display a page associated with the function.

SQL to display post

SELECT "Home_posts"."id", "Home_posts"."user_id", "Home_posts"."title", "Home_posts"."content", "Home_posts"."image", "Home_posts"."publish" FROM "Home_posts" WHERE "Home_posts"."id" = %d'

The get_absolute_url method is a method to get the URL of the post that a user is viewing to refresh the page found in Home/models.py under the Post model:

```
def get_absolute_url(self):
    return reverse('PostDetail', args=[str(self.id)])
    # returns string that can refer to post via http url
```

The total_likes method is a method to get the total number of users who have liked an individual post found in Home/models.py under the Post model:

```
def total_likes(self):
    return self.likes.count()
    # counts number of likes for each post
```

In Home/forms.py

```
class CommentForm(forms.ModelForm):
    # creates comment form

    class Meta:
        model = Comment
        # connects to comment table in database
        fields = ('content',)
        # field accessed is content

    def clean(self):
        # create method to clean data
        cleaned_data = super().clean()
        # store cleaned data in a variable
        content = cleaned_data.get("content")
        # get cleaned data from form
        with open("Home/banned words.txt") as file:
            # opens banned words as a text file
            lines = [line.strip() for line in file]
            # appends words to text list
        for word in lines:
            # if word in list
            if (word in content):
                raise forms.ValidationError(
                    "Banned word was used. Please remove the word '{}' and try again".format(word)
                )
            # raise validation error that banned word is in field
```

This displays the comments form with content as a text field which is acquired form the Comment table in the database imported in Home/forms.py from Home/models.py. This checks if there is any banned words in the field and raises a validation error if there are. This inherits the attributes and methods of the model specified in class Meta and the fields specified are the fields that will be used in the form.

SQL to comment on post

INSERT INTO "Home_comment" ("post_id", "user_id", "content", "publish") VALUES (%d, %d, \'%s\', \'%d\')

UPDATE "Home_comment" SET "post_id" = %d, "user_id" = %d, "content" = \'%s\', "publish" = \'%d\' WHERE "Home_comment"."id" = %d

In Home/urls.py

```
path('Posts/<int:pk>/', views.PostDetail, name='PostDetail'),
```

This links the URL Posts/<int: pk> with the post_detail.html page and PostDetail function in Home/views.py

If a user likes/dislikes the post:

In Home/views.py[65]

```
def LikePost(request):
    # creates function labeled LikePost with parameter request
    post = get_object_or_404(Post_model, id=request.POST.get('post_id'))
    # get post that is being liked by the user
    is_liked = False
    # is_liked is initially false
    if post.likes.filter(id=request.user.id).exists():
        # if the user already likes the post
        post.likes.remove(request.user)
        # remove the user from the posts likes
        is_liked = False
    else:
        post.likes.add(request.user)
        # add user to the posts likes
        is_liked = True
        # update boolean value
    return HttpResponseRedirect(post.get_absolute_url())
    # refresh the page with the post that the user is on
```

This will get the post a user is viewing and will add a like to the post if the user clicks the like button and hasn't liked the post or will remove the user if the user clicks the dislike button on the post. However, the user can only dislike the post to remove the post from being liked by the user previously. Get_object_or_404 is a method imported in Home/views.py to check if an object in a database table exists, if the object does not exist then a 404 page is displayed. HttpResponseRedirect is imported in Home/views.py to display a URL page after a form is submitted. Post_model refers to the Posts table in the database.

[65] https://www.youtube.com/watch?v=VoWw1Y5qqt8&list=WL&index=3&t=0s

SQL to like post

SELECT (1) AS "%s" FROM "auth_user" INNER JOIN "Home_posts_likes" ON ("auth_user"."id" = "Home_posts_likes"."user_id") WHERE ("Home_posts_likes"."posts_id" = %d AND "auth_user"."id" = %d) LIMIT 1

SQL to unlike post

SELECT (1) AS "%s" FROM "auth_user" INNER JOIN "Home_posts_likes" ON ("auth_user"."id" = "Home_posts_likes"."user_id") WHERE ("Home_posts_likes"."posts_id" = %d AND "auth_user"."id" = %d) LIMIT 1

DELETE FROM "Home_posts_likes" WHERE ("Home_posts_likes"."posts_id" = %d AND "Home_posts_likes"."user_id" IN (%d))

In Home/urls.py

```
path('Like', views.LikePost, name='LikePost'),
```

This links the URL Like to the LikePost function in Home/views.py

If a user wants to delete their post:

In Home/views.py[66]

```
def DeletePost(request):
    # create function DeletePost with parameter request
    post = get_object_or_404(Post_model, id=request.POST.get('post_id'))
    # get post to delete
    post.delete()
    # delete post from posts table
    return HttpResponseRedirect('../Posts')
    # redirect the user to the Posts view
```

This gets the object to delete from the objects id passed in the request to find the object in the Posts table. If the object exists the post is deleted and the user is redirected to the Posts page. Get_object_or_404 is a method imported in Home/views.py to check if an object in a database table exists, if the object does not exist then a 404 page is displayed. HttpResponseRedirect is imported in Home/views.py to display a URL page after a form is submitted. Post_model refers to the Posts table in the database.

SQL to delete post

SELECT "Home_posts"."id", "Home_posts"."user_id", "Home_posts"."title", "Home_posts"."content", "Home_posts"."image", "Home_posts"."publish" FROM "Home_posts" WHERE "Home_posts"."id" = %d'

DELETE FROM "Home_posts" WHERE "Home_posts"."id" IN (%d)

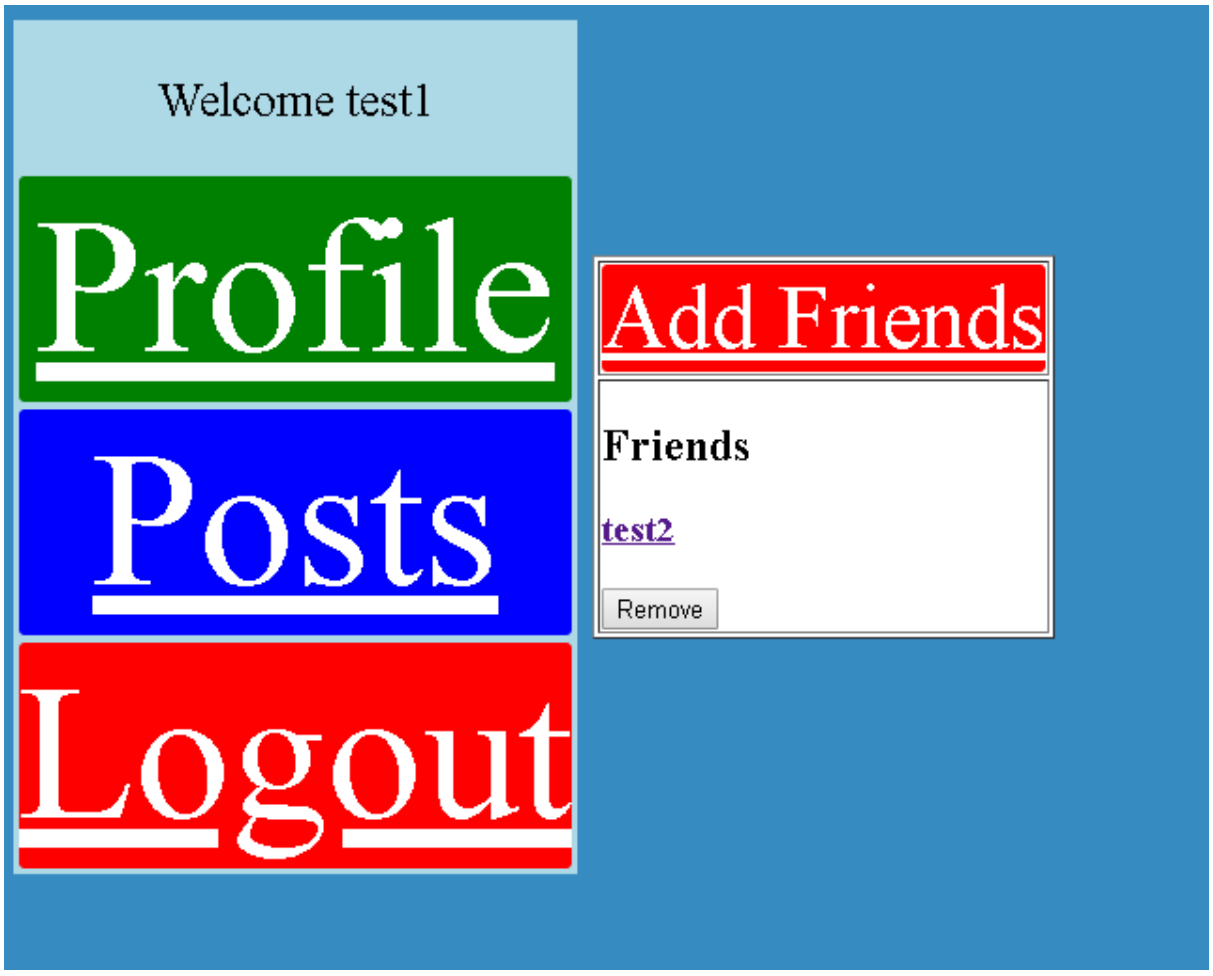
[66] https://www.youtube.com/watch?v=zwPsFIVMRX0&list=WL&index=13

In Home/urls.py

```
path('Delete', views.DeletePost, name='DeletePost'),
```

This links the URL Delete to the DeletePost function in Home/views.py

Friends



This will display the friends the user has which will allow users to view the profile of other users, remove users from their friends list, view a page to add friends, to view their own profile, view posts and logout of the social network.

HTML in Friends.html

```
{% extends 'Home/base2.html'%}
{#extends the base2.html page. This shows the relationship with the parent page base2.html and this child page#}
{% load static %}
{#django funciton to load css styles for page#}
{% block title %}Friends{% endblock %}
{#title tag to store title of page#}
{% block css %}<link rel="stylesheet" type="text/css" href="{% static 'css/Friends.css' %}">{% endblock %}
{#link to the css file to load the style for this page#}
{% block body %}
{#blocks for body to link to body elements in parent page.#}
{% if user.is_authenticated %}
{#if user is authenticated#}
<div class="container">
  <table>
    <tbody>
      <tr>
        <td>
      <table bgcolor = lightblue>
        <tbody>
        <tr>
          <td>
            <p align = "center"> Welcome {{user.username}} </p>
            {#display welcome next to users username#}
            <a href="{% url 'Profile' %}" class="Button1">Profile</button>
              {#button linking to profile page#}
          </td>
        </tr>
        <tr>
          <td>
            <a href="{% url 'Posts' %}" class="Button2">Posts</button>
            {#button linking to Posts page#}
          </td>
        </tr>
        <tr>
          <td>
            <a href="{% url 'Logout' %}" class="Button3">Logout</button>
              {#button to log out user#}
          </td>
        </tr>
```

```
        </tbody>
      </table>
      </td>
        <td>
      </td>
        <td>
      <table bgcolor = white border=black>
        {#in a table with a white backgorund#}
        <tbody>
          {%block content%}
            <tr>
              <td>
              <a href="{% url 'AddFriend' %}" class="Button4">Add Friends</button>
                {#button to add friends with link to AddFriend function#}
              </td>
            </tr>
            <tr>
              {%if friends%}
              {#if the user has friends#}
                <td>
                <h2>Friends</h2>
                {#friends in heading tags#}
                {%for user in friends%}
                {#for each user in the friends list#}
                  <a href="{% url 'ViewOtherProfile' pk=user.pk%}"><h3>{{user}}</h3></a>
                  {#display username and link to profile of user#}
                  <a href="{% url 'ChangeFriendStatus' operation='remove' pk=user.pk%}"><button type="button">Remove</button></a>
                  {#display button to remove user from friedns list linking to ChangeFriendStatus funciton#}
                {%endfor%}
                </td>
              {%endif%}
            </tr>
          {%endblock%}
        </tbody>
      </table>
      </td>
      </tr>
    </tbody>
  </table>
```

```
  {% else %}
  <p>You can't access this page as you aren't logged in</p>
  {#display message if user is not logged in#}
  {% endif %}
</div>
{% endblock %}
```

In Home/views.py[67]

```
def Friends(request):
    # create function labeled Friends with parameter request
    try:
        friend = Friend.objects.get(current_user=request.user)
        friends = friend.users.all()
        # try to get the current users friends list
    except Friend.DoesNotExist:
        friends = None
        # unless the current user has no friends
    context = {
        'friends': friends
    }  # store the friends in a context variable to use on the template
    return render(request, 'Home/Friends.html', context)
    # render the template Friends.html with the friends list of the user
```

This displays all the friends the current user has and creates a list of the users that have a friend relationship with the current user to view on the current users' friends' page. I have used exception handling as running the query will produce an error if the user has no friends thus if the user has

[67] https://www.youtube.com/watch?v=ZGbYmMF4QLI&list=WL&index=6

friends get a list of all the friends except the friends object does not exist then return None in the friends list. Friend refers to the Friends table in the database imported in Home/views.py from Home/models.py. Render is imported in Home/views.py to display a page associated with the function.

SQL to display current friends

SELECT "Home_friend"."id", "Home_friend"."current_user_id" FROM "Home_friend" WHERE "Home_friend"."current_user_id" = %d

SQL to remove friend

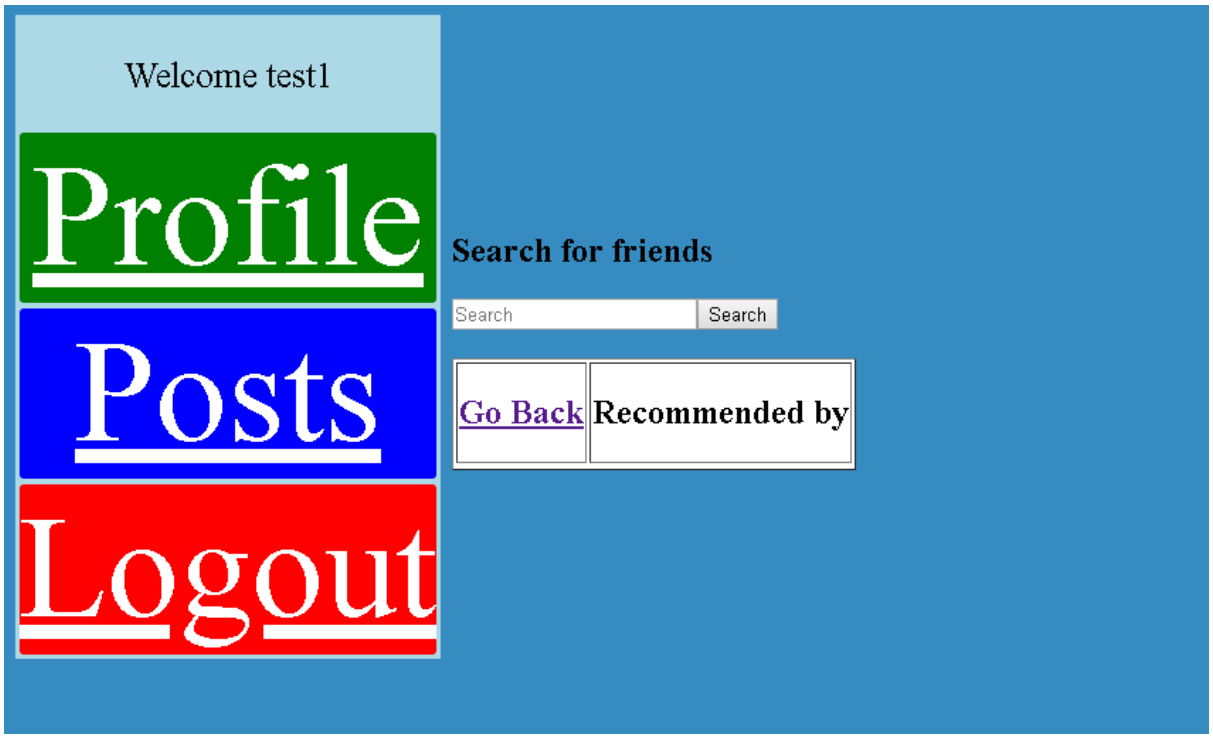
DELETE FROM "Home_friend_users" WHERE ("Home_friend_users"."friend_id" = %d AND "Home_friend_users"."user_id" IN (%d))

In Home/urls.py

```
path('Friends', views.Friends, name='Friends'),
```

This connects the URL Friends to the function Friends in Home/views.py and the Friends.html page.

AddFriend[68]



This page allows users to add friends based on year group, post code, interest and town. This also allows users to search for friends to add, go back to view their friends list, go to their profile page, go to another users profile page, and go to the posts page and logout.

[68] https://www.youtube.com/watch?v=wLZlyjq_mFc&list=WL&index=7

HTML in AddFriends.html

```
{% extends 'Home/Friends.html'%}
{#extends the Friends.html page. This shows the relationship with the parent page Friends.html and this child page#}
{%block content%}
<h2>Search for friends</h2>
{#search for friends in heading tags#}
<td>
  <a class="Button" href = "{% url 'Friends' %}"><h2>Go Back</h2></a>
  {#button to go back to their friends list page#}
</td>
<td>
<h2>Recommended by</h2>
{#recommended by in heading tags#}
</td>
<tr>
  <form method="GET" action="{% url 'SearchFriends' %}">
    <input name="query" value="{{request.GET.query}}" placeholder="Search">
    <button type="submit">Search</button>
    <br><br>
    {#search form to search friends#}
  {%if yearlist%}
  {#if users in same year#}
    <td>
    <h2>YearGroup</h2>
    {%for user in yearlist%}
      <a href="{% url 'ViewOtherProfile' pk=user.pk%}"><h3>{{user}}</h3></a>
      {#display users with link to their profile page#}
      {% if not user in friends %}
      {#if the users arent friends#}
        <a href="{%url 'ChangeFriendStatus' operation='add' pk=user.pk%}"><button type="button">Add</button></a><br>
        {#display button to add friend#}
      {%endif%}
    {%endfor%}
    </td>
  {%endif%}
  {%if interestlist%}
  {#if users have same interest#}
    <td>
    <h2>Interests</h2>
    {% for user in interestlist %}
```

```
      <a href="{% url 'ViewOtherProfile' pk=user.pk%}"><h3>{{user}}</h3></a>
      {#display users with link to their profile page#}
      {% if not user in friends %}
      {#if the users arent friends#}
        <a href="{%url 'ChangeFriendStatus' operation='add' pk=user.pk%}"><button type="button">Add</button></a><br>
        {#display button to add friend#}
      {%endif%}
    {% endfor %}
    </td>
  {%endif%}
  {%if postcodelist%}
  {#if users have same postcode#}
    <td>
    <h2>Postcode</h2>
    {% for user in postcodelist %}
      <a href="{% url 'ViewOtherProfile' pk=user.pk%}"><h3>{{user}}</h3></a>
      {#display users with link to their profile page#}
      {% if not user in friends %}
      {#if the users arent friends#}
        <a href="{%url 'ChangeFriendStatus' operation='add' pk=user.pk%}"><button type="button">Add</button></a><br>
        {#display button to add friend#}
      {%endif%}
    {% endfor %}
    </td>
  {%endif%}
  {%if townlist%}
  {#if users in same town#}
    <td>
    <h2>Town</h2>
    {% for user in townlist %}
      <a href="{% url 'ViewOtherProfile' pk=user.pk%}"><h3>{{user}}</h3></a>
      {#display users with link to their profile page#}
      {% if not user in friends %}
      {#if the users arent friends#}
        <a href="{%url 'ChangeFriendStatus' operation='add' pk=user.pk%}"><button type="button">Add</button></a><br>
        {#display button to add friend#}
      {%endif%}
    {% endfor %}
    </td>
```

```
  {%endif%}
</tr>
{%endblock%}
```

In Home/views.py[69] [70]

```
def AddFriend(request):
    # create function labeled AddFriend with parameter request
    user = request.user
    # store current user as user
    try:
        friend = Friend.objects.get(current_user=request.user)
        friends = friend.users.all()
        # try to get the current users friends list
    except Friend.DoesNotExist:
        friends = None
        # unless the current user has no friends
    superuser = User.objects.get(username="Kamran")  # store superuser as admin
    yearlist = UserInformation.objects.filter(year=user.userinformation.year).exclude(user=user)
    yearlist = yearlist.exclude(user=superuser)
    interestlist = UserInformation.objects.filter(interest=user.userinformation.interest).exclude(user=user)
    interestlist = interestlist.exclude(user=superuser)
    postcodelist = UserInformation.objects.filter(postcode=user.userinformation.postcode).exclude(user=user)
    postcodelist = postcodelist.exclude(user=superuser)
    townlist = UserInformation.objects.filter(town=user.userinformation.town).exclude(user=user)
    townlist = townlist.exclude(user=superuser)
    # exclude admin and current user from all friends lists that will be displayed
    context = {
        'yearlist': yearlist,
        'interestlist': interestlist,
        'postcodelist': postcodelist,
        'townlist': townlist,
        'friends': friends
    }  # pass friends lists as context variables in template
    return render(request, 'Home/AddFriends.html', context)
    # render template AddFriends.html with each friends list
```

This uses exception handling to view if the user already has friends and if the current user has no friends, the variable passed into the template is None. Then the UserInformation table is filtered to find users of the same interests, year group, town, postcode whilst excluding the current user and super user from the list to be presented to the user as recommended friends' lists. Friend refers to the Friends table in the database imported from Home/models.py in Home/views.py. User refers to the User table in the database imported from Django's authenticated models in Home/views.py UserInformation refers to the UserInformation table in Home/models.py imported in Home/views.py. Render is imported in Home/views.py to display a page associated with the function.

SQL to exclude super user from friends list

SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user" WHERE "auth_user"."username" = \'Kamran\'

SQL to exclude current user from friends list

SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user" WHERE "auth_user"."id" = %d

[69] https://www.youtube.com/watch?v=bFhuOULgKDs&list=WL&index=8
[70] https://www.youtube.com/watch?v=IXJ46DitsIg&list=WL&index=9

SQL to add friend

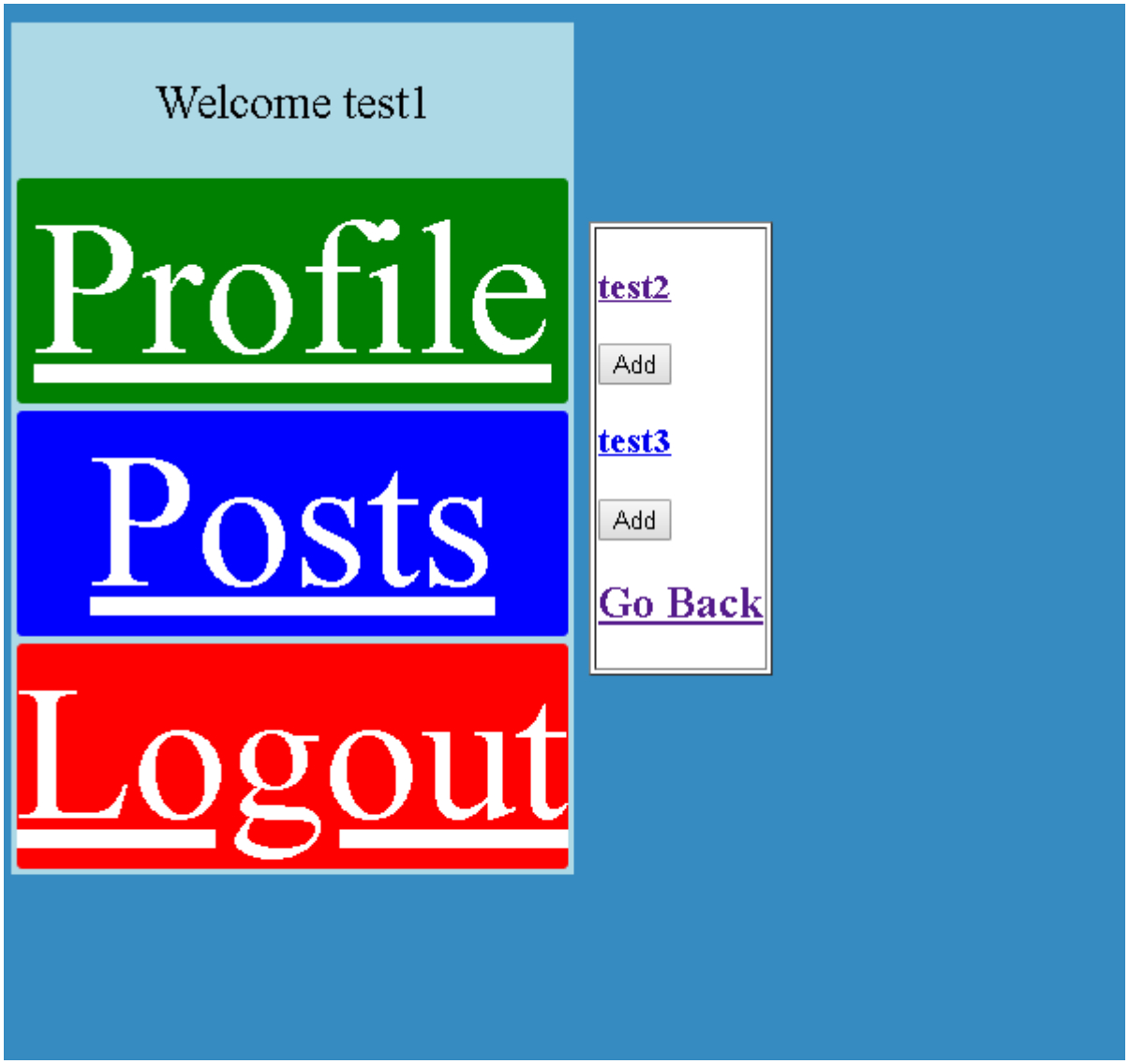
INSERT INTO "Home_friend_users" ("friend_id", "user_id") SELECT %d, %d

In Home/urls.py

```
path('AddFriends', views.AddFriend, name='AddFriend'),
```

This links the URL AddFriends to the AddFriends.html page and AddFriend function in Home/views.py

If a user searches to add friends:



This displays the searched users based on the users' username, first name and last name. From this page a user can add users to their friends list, go back to view their friends list, go to their profile page or other users profile page, go to the posts page and logout.

HTML in friend search.html

```
{% extends 'Home/Friends.html'%}
{#extends the Friends.html page. This shows the relationship with the parent page Friends.html and this child page#}
{%block content%}
      <tbody>
        <td>
          {%if results%}
          {#if there are results to show#}
              {% for user in results %}
                <a href="{% url 'ViewOtherProfile' pk=user.pk%}"><h3>{{user}}</h3></a>
                {#display each username and a link to their profile#}
                {% if not user in friends %}
                {#if the users arent friends#}
                  <a href="{%url 'ChangeFriendStatus' operation='add' pk=user.pk%}"><button type="button">Add</button></a><br>
                  {#display an add friend button#}
                {%endif%}
              {% endfor %}
          {%else%}
            <h1>There are no users with that username, firstname or lastname</h1>
            {#else display this error message#}
          {%endif%}
          <a class="Button" href = "{% url 'Friends' %}"><h2>Go Back</h2></a>
          {#link to go back to view the users friends list#}
        </td>
      </tbody>
{%endblock%}
```

In Home/views.py

```
def SearchFriends(request):
    # create search friends function with parameter request
    query = request.GET.get('query')
    # store search parameters as a query
    if query:
        # if a query has been made when searching
        superuser = User.objects.get(username="Kamran")
        # define super user as the admin
        results = User.objects.filter(Q(username__icontains=query) | Q(first_name__icontains=query) | Q(last_name__icontains=query))
        # search for users with the query in the username, firstname, lastname
        results = results.exclude(id=request.user.pk)
        results = results.exclude(id=superuser.pk)
        # exclude the current user and super user from the results
    else:
        results = False
        # return false if no query was made
    context = {
        "results": results,
    }  # pass results as context variable in template
    return render(request, "Home/friend search.html", context)
    # render remplate friend search.html with results of search
```

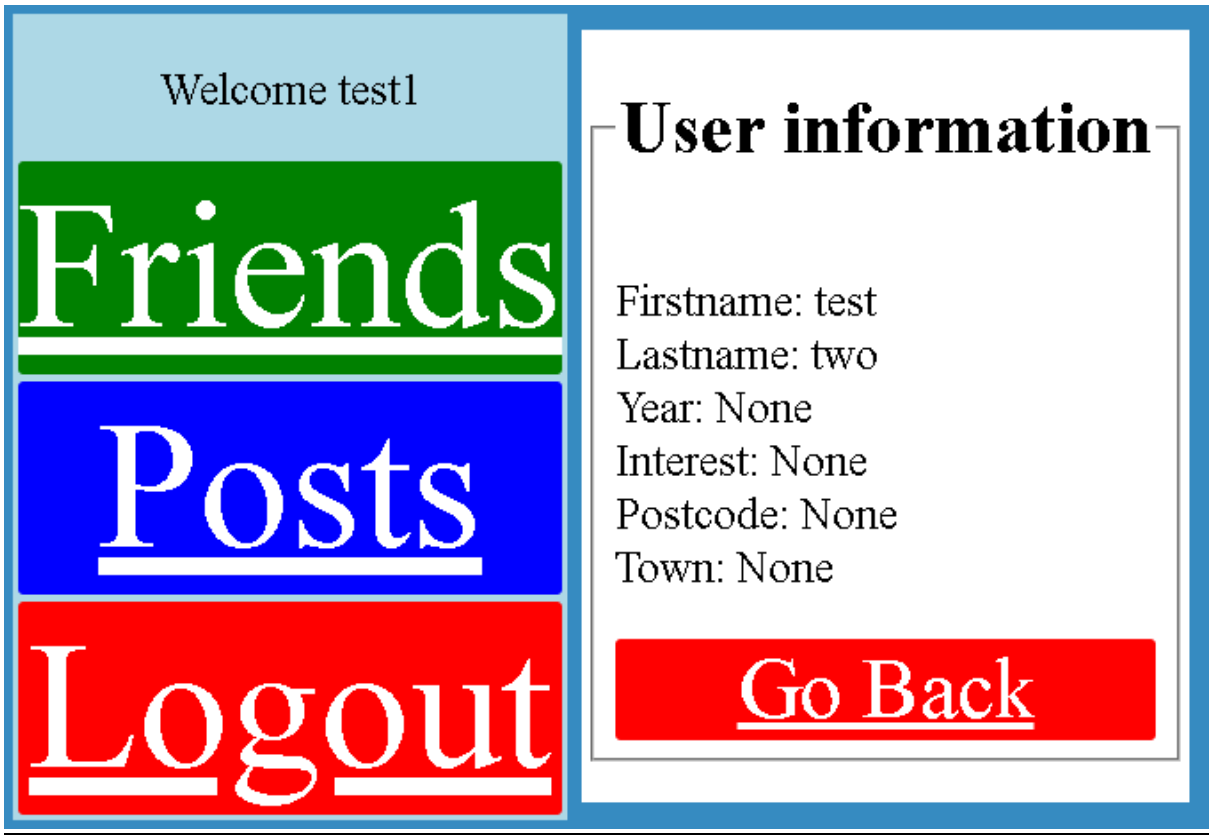
This retrieves the query submitted in the search field on the Add Friends page and filters the objects in the User table to locate if there are any users with that username or first name or last name otherwise an error message is displayed. User refers to the User table in the database from Home/models.py imported in Home/views.py. This excludes the administrator and current user from the results of the query. Q is a method imported in Home/views.py which will let me query the database with a query defined by a user in a form. Render is imported in Home/views.py to display a page associated with the function.

In Home/urls.py

```
path('AddFriends/results/', views.SearchFriends, name='SearchFriends')
```

This connects the AddFriends/results/ URL to the friend search.html page and SearchFriends function in Home/views.py

If a username is clicked on the Friends page/ AddFriends page or SearchFriends page:



This will display the users' profile that you can view. The buttons on the page allow the user to go back to the friends' page to view your friends or to view your profile, posts page or to logout.

SQL

SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login", "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name", "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff", "auth_user"."is_active", "auth_user"."date_joined" FROM "auth_user" WHERE "auth_user"."id" = %d

HTML in Profile.html

The original code on the profile page displays the users first name, last name, year, interest, postcode and town however the code below changes the edit profile button to a go back button to allow the user to go back to the friends page to view the logged in users friends.

```
<a href="{% url 'AddFriend' %}" class="Button4">Go Back</button>
  {#return to the AddFriend page as you can only view other profile pages from AddFriend#}
```

In Home/views.py

```
if pk:
    # if a pk is supplied
    user = User.objects.get(pk=pk)
    # get the users data for their profile
```

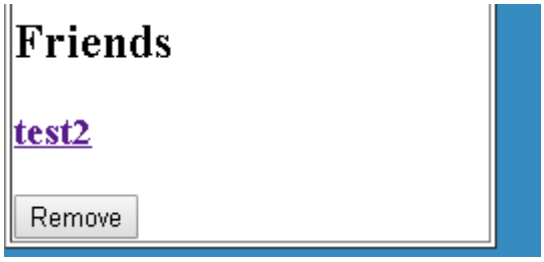
This code selects the user with the primary key passed through the request to use in displaying other user profiles. User references the User table in the database.

In Home/urls.py

```
path('Profile/<int:pk>', views.Profile, name='ViewOtherProfile'),
```

This connects the URL Profile/<int: pk> with the Profile.html page and the function Profile in Home/views.py. This displays the users' primary key in the profiles urls when displaying another user's profile.

Add/remove friends



This allows the user to add friends from the add friends page and search friends page but removing friends is only possible from the logged in users friends page.

HTML in Friends.html

```
<a href="{% url 'ChangeFriendStatus' operation='remove' pk=user.pk%}"><button type="button">Remove</button></a>
{#display button to remove user from friedns list linking to ChangeFriendStatus funciton#}
```

This passes the operation to remove the friend and primary key of the user to the function ChangeFriendStatus in Home/views.py.

HTML in friend search.html and AddFriends.html

```
<a href="{%url 'ChangeFriendStatus' operation='add' pk=user.pk%}"><button type="button">Add</button></a><br>
{#display an add friend button#}
```

This passes the operation to add the friend and primary key of the user to the function ChangeFriendStatus in Home/views.py.

In Home/views.py[71] [72]

```
def ChangeFriendStatus(request, operation, pk):
    # create function labeled ChangeFriendStatus with parameters request, operation and pk
    new_friend = User.objects.get(pk=pk)
    # store the passed primary key as new_friend variable
    if operation == "add":
        # if the operaiton is add
        Friend.make_friend(request.user, new_friend)
        # make a friend relationsihp between the logged in user and new_friend
    elif operation == "remove":
        # if the operation is remove
        Friend.lose_friend(request.user, new_friend)
        # remove the friend relationship between the logged in user and new_friend
    return redirect('Friends')
    # redirect to the Friends url
```

This creates or removes a friend relationship between a selected user and the current logged in user. User refers to the User table in the database imported from Home/models.py in Home/views.py. The new_friend stores the user selected by the logged in user. If the operation passed in the function is add, use the make_friend method to make a relationship between the current user and the selected user. If the operation passed in the function is remove, use the lose_friend method to remove the relationship between the current user and the selected user. Friend refers to the Friends table in the database from Home/models.py in Home/views.py. Redirect is a method imported in Home/views.py to redirect the user to the page associated with the Friends URL.

In Home/models.py under Friend

```
@classmethod
def make_friend(cls, current_user, new_friend):
    # method to make friends
    friend, created = cls.objects.get_or_create(
        current_user=current_user
    )
    friend.users.add(new_friend)
    # get or create object with owner of friendship and created to add friend
```

This is the class method called when the operation is add. This creates or gets the users friends object in the Friends table. This will create a friend relationship between the current user who is logged in and the new_friend which is selected by the logged in user.

```
@classmethod
def lose_friend(cls, current_user, new_friend):
    # method to lose friend
    friend, created = cls.objects.get_or_create(
        current_user=current_user
    )
    friend.users.remove(new_friend)
    # get or create object with owner of freinship and created to remove friend
```

[71] https://www.youtube.com/watch?v=nwpLCa79DUw&list=WL&index=10
[72] https://www.youtube.com/watch?v=zcJegVlKqqs&list=WL&index=11

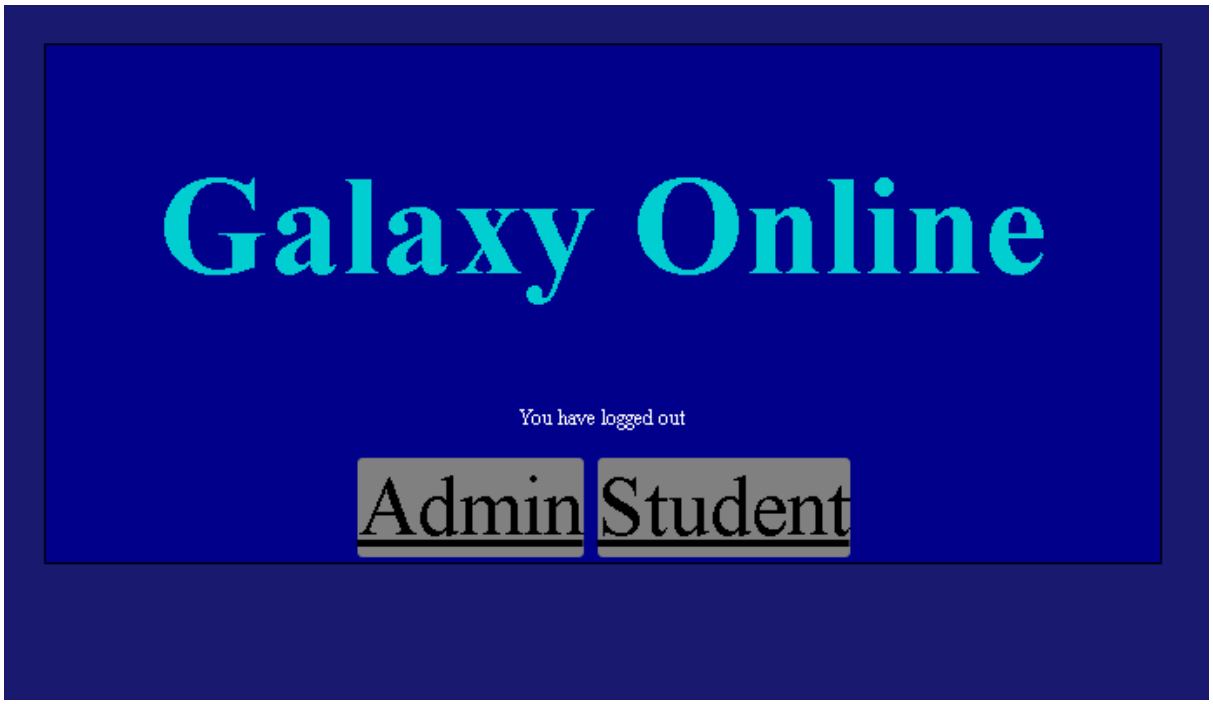
This is the class method called when the operation is remove. This creates or gets the users' friends object in the Friends table. This will remove a friend relationship between the current user who is logged in and the new_friend which is selected by the logged in user.

In Home/urls.py

```
path('connect/<operation>/<int:pk>', views.ChangeFriendStatus, name='ChangeFriendStatus'),
```

This stores the operation that is passed when selecting to add or remove a friend and the primary key of the user which is placed in <operation> and <int: pk> respectively. This URL links to the ChangeFriendStatus function in Home/views.py.

Logout



This page is displayed when a student logs out of the social network from any page on the social network. This will allow another user to choose to login to the admin section of the site or student section of the site.

HTML in logout.html

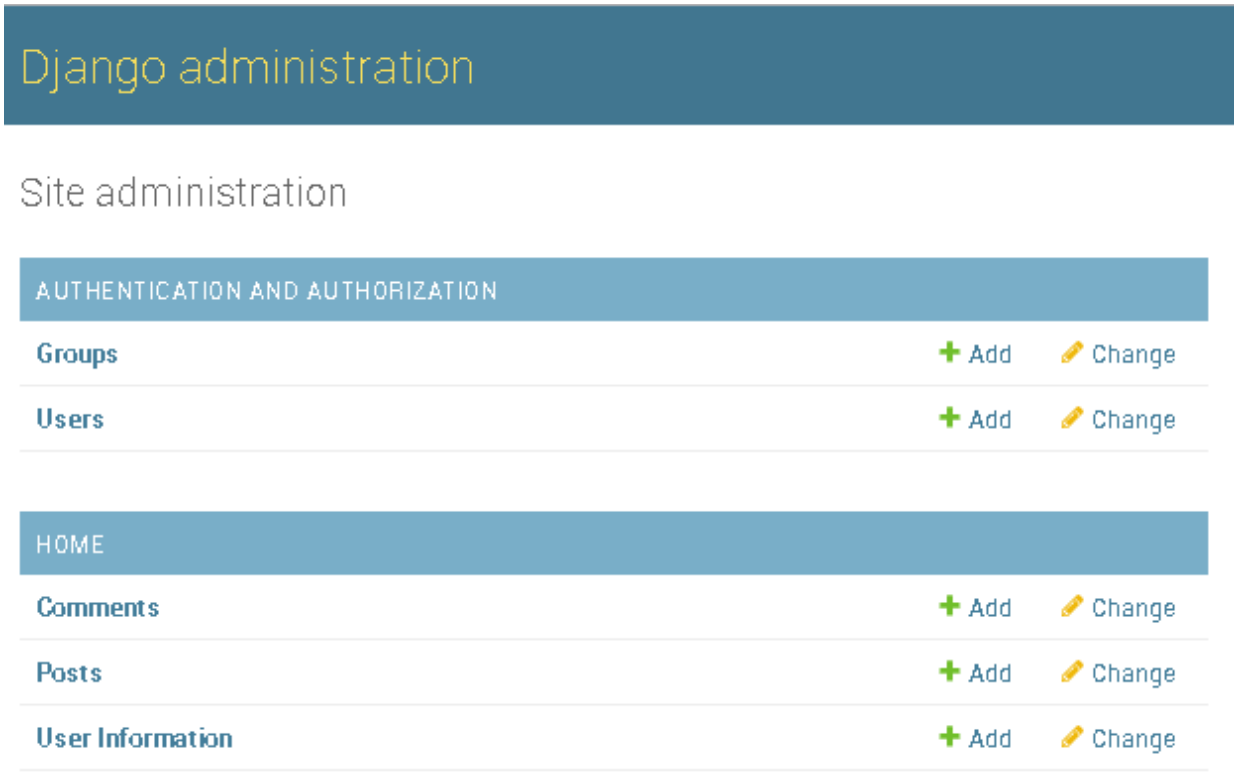
```
{%extends "Home/index.html"%}
{#extends the index.html page. This shows the relationship with the parent page index.html and this child page#}
{% block content%}
<p> You have logged out </p>
{#displays logout text#}
{% endblock %}
```

In Home/urls.py

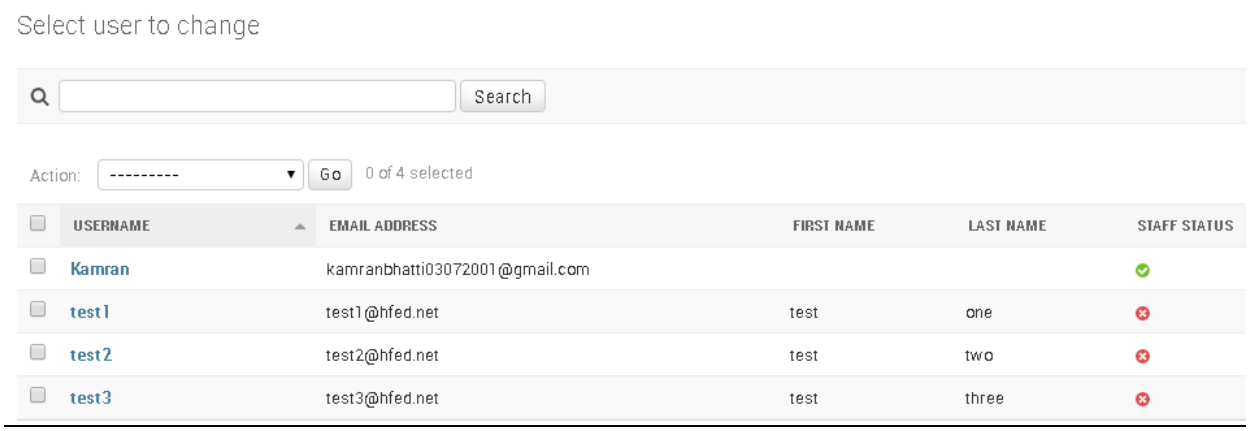
```
path('Logout', auth_views.LogoutView.as_view(template_name='Home/logout.html'), name='Logout'),
```

This links the URL Logout to the authenticated view for the logout view supplied by a Django module however, I have passed the template I have created being logout.html to display as the logout screen for students who are logging out of the social network.

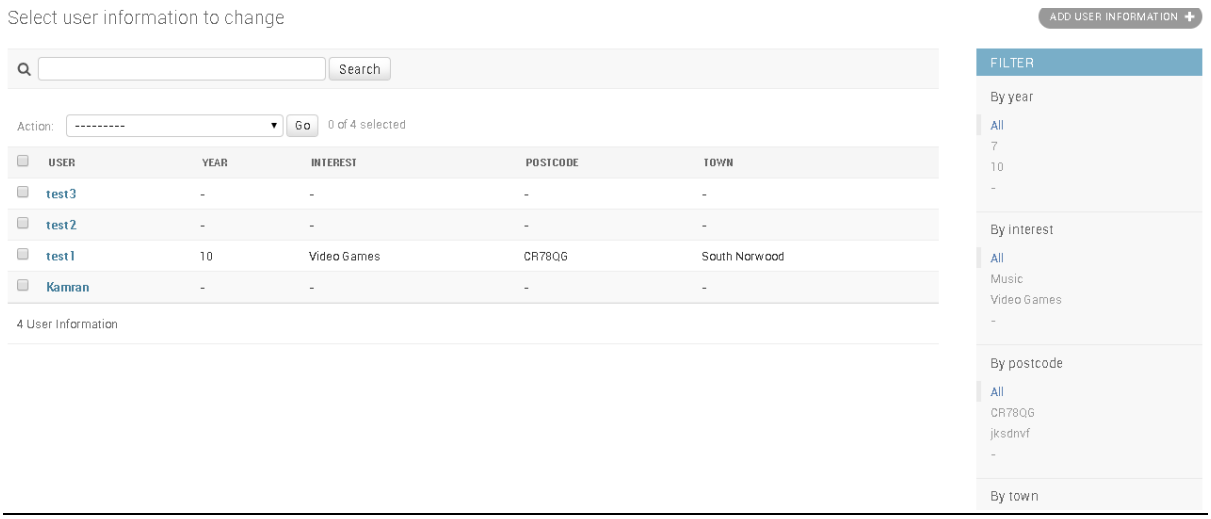
Admin



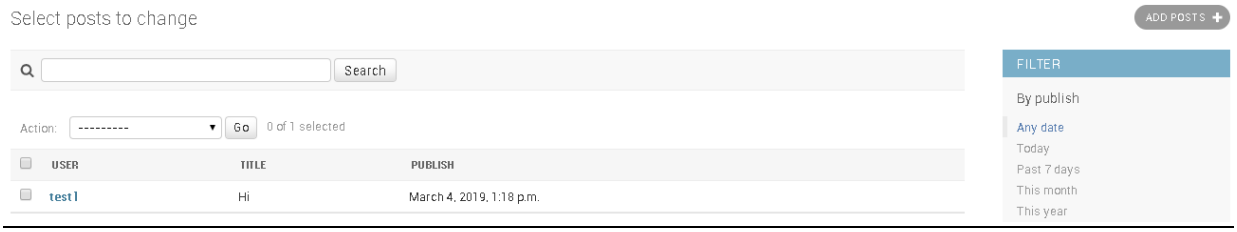
Users table in Admin view:



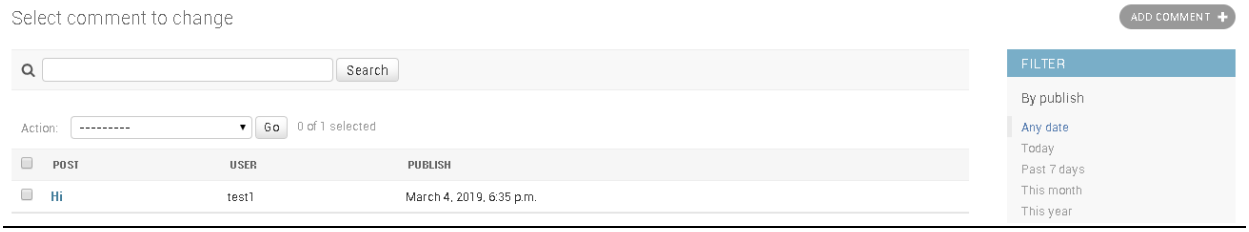
UserInformation table in Admin view:



Posts table in admin view:



Comments table in admin view:



After successfully logging into the administration section of the website, the administrator is able to view the users on the social network, create new users, delete users, view comments for posts on the social network, create comments on posts, delete comments on posts, create posts, view posts made by students, delete posts and view or delete each students user information on the social network.

In Home/models.py

Under YearGroup

```
def __str__(self):
    return str(self.year)
    # returns string value of field in table
```

This will display the year object that is associated to each user as a string instead of an integer or primary key value when being viewed in Django admin.

Under Interests

```
def __str__(self):
    return self.interest
    # returns string value of field in table
```

This will display the interest object that is associated with each user as a string instead of a primary key when being viewed in Django admin.

Under Postcode

```
def __str__(self):
    return self.postcode
    # returns string value of field in table
```

This will display the postcode object that is associated with each user as a string instead of a primary key when being viewed in Django admin.

Under Town

```
def __str__(self):
    return self.town
    # returns string value of field in table
```

This will display the postcode object that is associated with each user as a string instead of a primary key when being viewed in Django admin.

Under User Information

```
class Meta():
    # specifies customisation to table
    verbose_name_plural = 'User Information'
    # gives a pluralised name to the database table in django admin

def __str__(self):
    return self.user.username
    # returns string value of users username in table
```

This will display the User Information object that is associated with each user as the users' username instead of a primary key or object instance when being viewed in Django admin. Additionally this will

show the pluralised name of User Information as User Information when multiple users are signed up to the social network.

Under Posts

```
class Meta():
    # specifies customisation to table
    verbose_name_plural = 'Posts'
    # gives pluralised name to the database table in django admin

def __str__(self):
    return self.title
    # returns string value of field in table
```

This will display the post object that is associated with each user as the posts title instead of a primary key when being viewed in Django admin. Additionally this will show the pluralised name of Posts as Posts when multiple posts are in the database.

Under Comment

```
def __str__(self):
    return "{}-{}".format(self.post.title, str(self.user.username))
    # returns string value of the posts title and users username in table
```

This will display the comment object that is associated with each user as a string with the posts title and the user which commented on the post instead of a primary key when being viewed in Django admin.

In Home/admin.py[73]

```
from django.contrib import admin
# imports admin method
from .models import UserInformation, Posts, Comment
# imports all of the tables i want the admin to have access to


class UserInformationAdmin(admin.ModelAdmin):
    list_display = ('user', 'year', 'interest', 'postcode', 'town')
    search_fields = ['user__username', 'user__first_name', 'user__last_name', 'user__email']
    list_filter = ['year', 'interest', 'postcode', 'town']


class PostsAdmin(admin.ModelAdmin):
    list_display = ('user', 'title', 'publish')
    search_fields = ['user__username', 'user__first_name', 'user__last_name', 'user__email', 'title', 'content']
    list_filter = ['publish']


class CommentAdmin(admin.ModelAdmin):
    list_display = ('post', 'user', 'publish')
    search_fields = ['post__title', 'post__content', 'user__username', 'user__first_name', 'user__last_name', 'user__email', 'content']
    list_filter = ['publish']

# above lines specify which fields to display in django admin
# and what the search field is specifically searching in and filters when displaying the records

admin.site.register(UserInformation, UserInformationAdmin)
admin.site.register(Posts, PostsAdmin)
admin.site.register(Comment, CommentAdmin)
# registers specified database tables on the admin site for the admin to interact with
```

[73] https://stackoverflow.com/questions/10543032/how-to-show-all-fields-of-model-in-admin-page

This displays the UserInformation, Posts and Comments tables which I have created in my database in Home/models.py which I have imported in Home/admin.py. The User table is already imported in Django admin thus I did not need to register the table to view on the admin site. This specifies that each record in the UserInformation table should display the users' username, year, postcode, town and interest and for the search parameter to be by the users username and to filter the records by year, interest, postcode and town. Additionally, each record in the Posts table should display the title, user who created the post and the date and time of when the post was published with search field for the users' username, title of post and content in post with filters for the time when the post was published. Furthermore each record in the Comments table should display the post the comment is assigned to, the user who commented and the date and time of the comment with search for the posts title, users' username and content in the comments and filtering by the date and time of publishing the comment.

In GalaxyOnline/urls.py

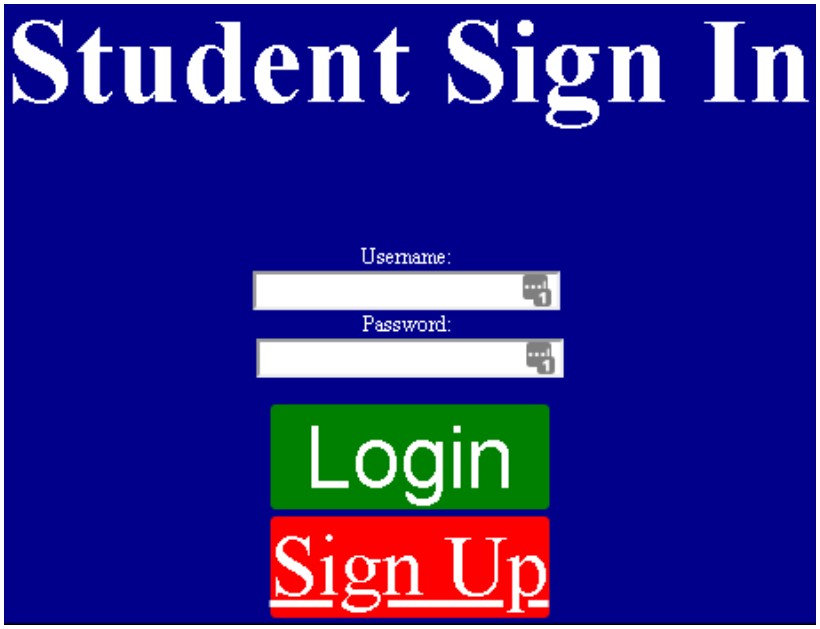
```
path('admin/', admin.site.urls),
```

This connects the admin/ URL to the admin.site.urls which was automatically generated by Django to be used in means of accessing administrative tools from the website which I will be using to edit tables in my database as an admin.

TESTING

Test 1

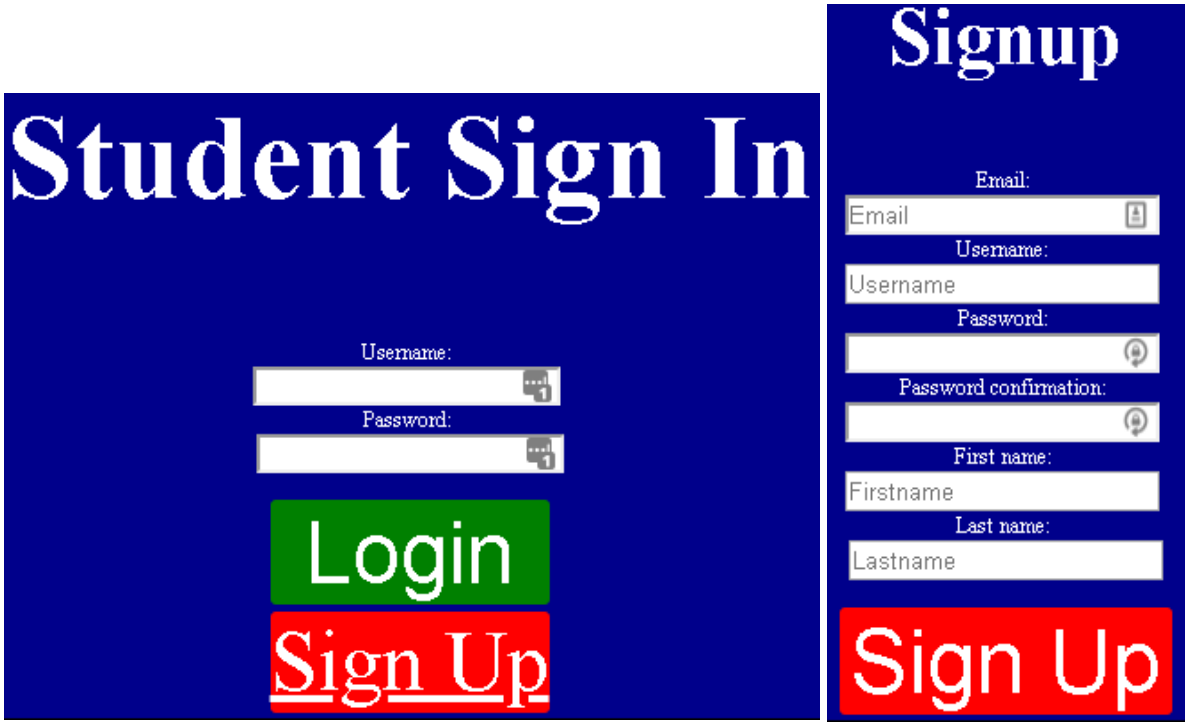
Clicking student will take a user to the student login page from the index page which is a typical data type. The expected result is to be shown the student login page after clicking the student button.



Result: pass

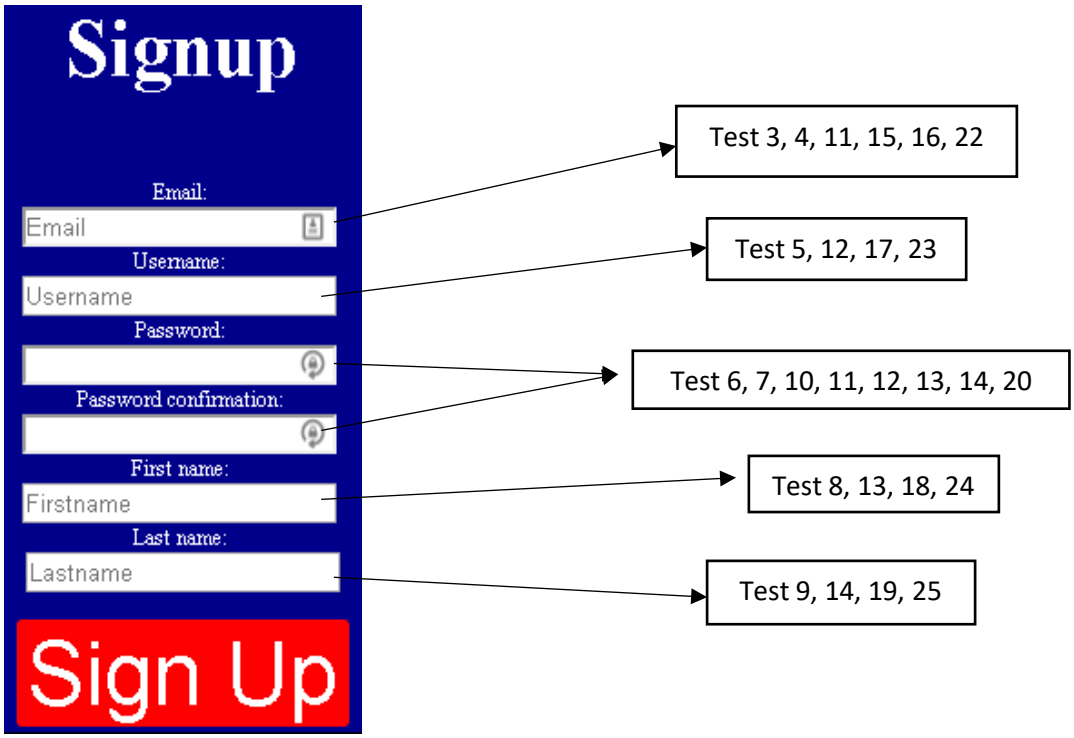
Test 2

Clicking the signup button will take a user to the signup page from the student sign in page. This is a typical data type. The expected result is to be shown the sign up page after clicking the sign up button.



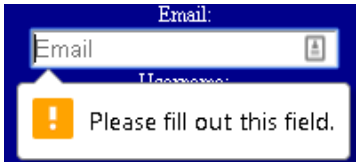
Result: pass

Signup Page



Test 3

Clicking the signup button on an empty form will not submit. Data type is typical. Expected result is for a pop up to show on fields if fields are not filled



Results: pass

Test 4

Clicking signup after a value is entered in email will display an error message if the email doesn't have a '@' symbol. Data type is typical. Expected result is that an error will display if the email field does not contain a '@' symbol.

Result: pass

Test 5

Whilst having the email field filled, if the username field is not filled on clicking sign up an error is raised. Data type is typical. Expected result is for an error to be displayed if the username field is not filled on submitting the form

Result: pass

Test 6

After filling in the username field trying to submit the form without filling in the first password field will raise an error. Data type is typical. Expected result is for an error to be raised if the form is attempted to be submit without filling in the first password field.

Result: pass

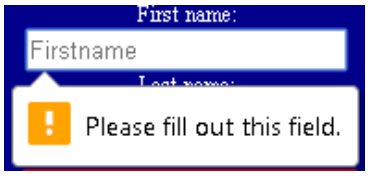
Test 7

If second password field is not filled when submitting the form, an error is raised. Data type is typical. Expected result is to see an error message.

Result: pass

Test 8

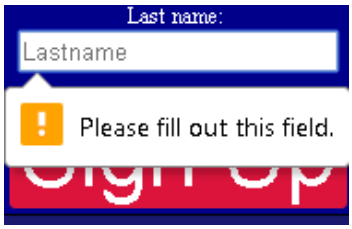
Testing that if first name is not filled when submitting form an error message is displayed. Data type is typical. Expected result is for an error message to be displayed.



Result: pass

Test 9

If last name is left blank after trying to submit the form an error message is displayed. Data type is typical. Expected result is for an error message to be displayed if the last name is not filled on trying to submit the form.



Result: pass

Test 10

If passwords do not match error is raised. Data type is typical. Expected result is to see an error.

Input in first password field: qwertyuiop input in second password field: asdfghjkl

Result: pass

Test 11

If password is similar to email error is raised. Data type is typical. Expected result is to see an error message. Password field: kamran5384. Email field: kamran@hfed.net

Result: pass

Test 12

If the password is similar to the username an error is raised. Data type is typical. Expected result is to see an error message. Password input: kamran5485, username field: kamran5384

Result: pass

Test 13

If password is too similar to first name an error is raise. Data type is typical. Password input is kamranmuhammad. First name is kamranmuhammad.

Result: pass

Test 14

If password is too similar to last name, error is raised. Data type is typical. Expected result is to see an error. Password input is kamranmuhammad. Last name is kamranmuhammad.

Result: pass

Test 15

If email is not an hfed email then an error is raised. Data type is erroneous. Expected result is to see an error message. Email is k.bhatti@gmail.com

Result: pass

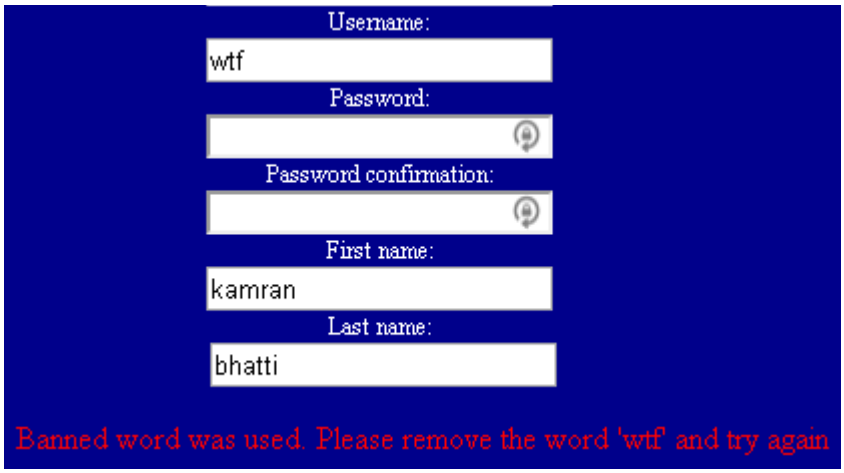
Test 16

If the email has a banned word in the field an error is raised. Data type is erroneous. Expected result is to see an error. Email is wtf@hfed.net

Result: pass

Test 17

If username has banned word in field error message is raised. Data type is erroneous, Expected result is to see an error. Username is wtf.



Result: pass

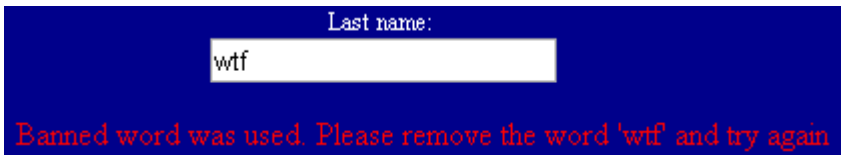
Test 18

If first name has banned word in field error message is raised. Data type is erroneous. Expected result is to see an error. First name is wtf.

Result: pass

Test 19

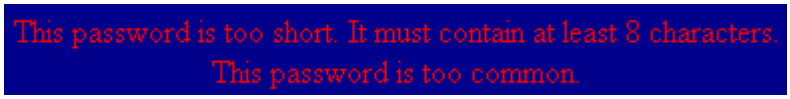
If last name has banned word in field error message is raised. Data type is erroneous. Expected result is to see an error. Last name is wtf.



Result: pass

Test 20

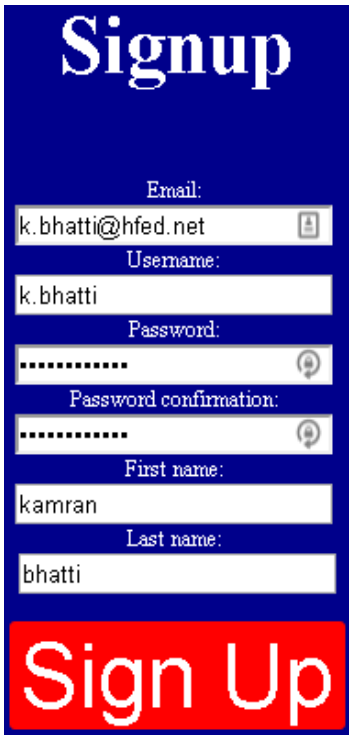
If password is too simple/common and too short error is raised. Data type is erroneous. Expected result is to see an error. Password input is a.

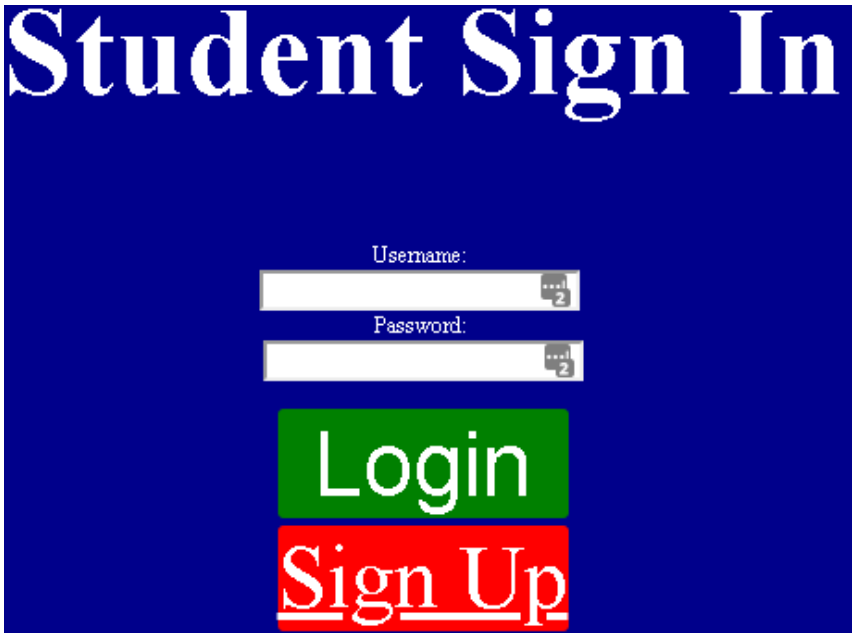


Result: pass

Test 21

If login form is valid, details are saved to the database and a user information relation is created and the user will be redirected to the login screen. Data type is typical. Expected result is for the new user to be added to the database. Username: k.bhatti, email: k.bhatti@hfed.net password: eJXFB76nJwwY, first name: kamran, last name: bhatti.





In User table:

12	pbk...	NULL	0	k.bhatti	kamran	k.bhatti@hfed.net	0	1	201...	bhatti

In user information table:

12	NULL	NULL	NULL	12	NULL

12 being the primary key of the user.

Result: pass

Test 22

If another user tries to sign up with the same email, an error is raised. Data type is typical. Expected result is to see an error. Email: k.bhatti@hfed.net

Result: pass

Test 23

If another user tries to sign up with the same username, an error is raised. Data type is typical. Expected result is to see an error. Username: k.bhatti

Result: pass

Test 24

If a user has a number in their first name, an error is raise. Data type is erroneous. Expected result is to see an error. First name: kamran1

Result: pass

Test 25

If a user has a number in their last name, an error is raised. Data type is erroneous. Expected result is to see an error. Last name: bhatti1

Result: pass

Student Login Page

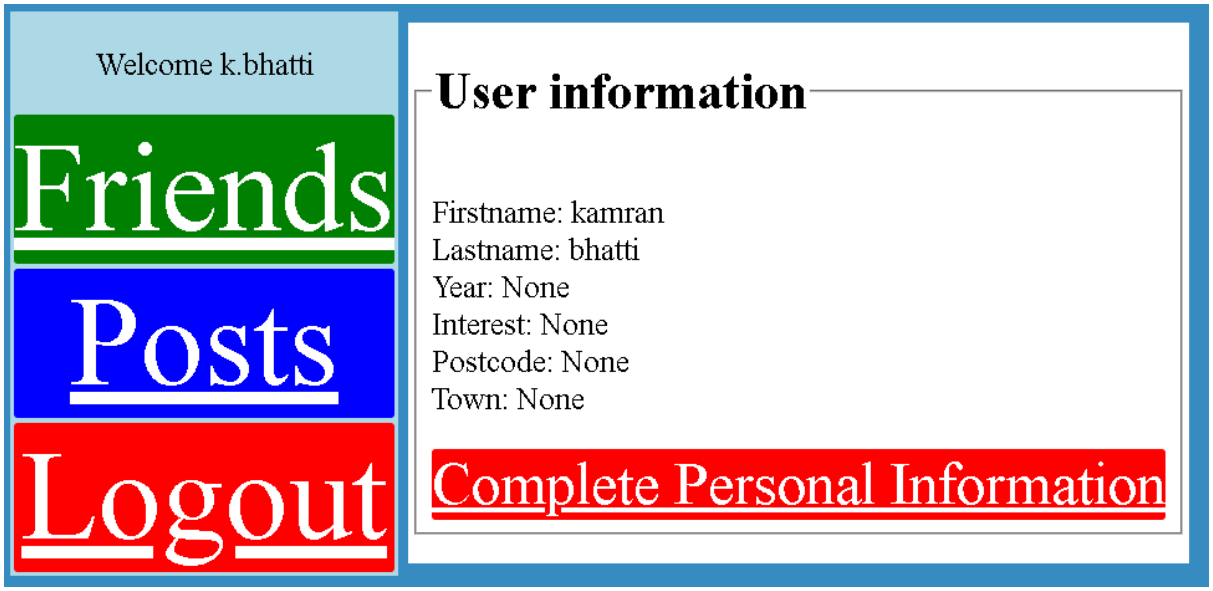
Test 26

If user details are incorrect an error is raised. Data type is typical. Expected result is to see an error.

Result: pass

Test 27

If user details are correct, user is redirected to profile page. Data type is typical. Expected result is to see the profile page after successfully logging in. username: k.bhatti password: eJXFB76nJwwY



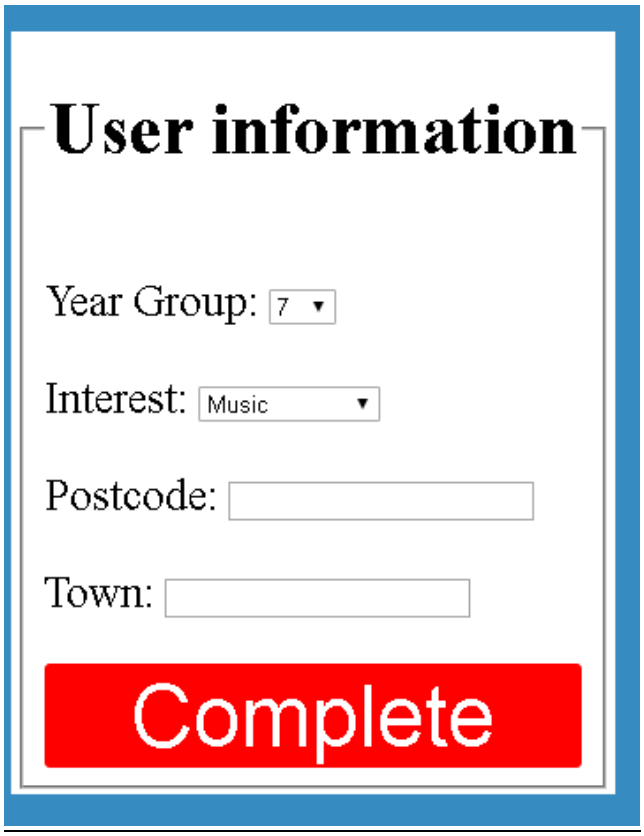
Result: pass

Test 28

Clicking the complete personal information button will allow me to edit the profile page. Data type is typical. Expected result is to be redirected to a form to edit the profile page.

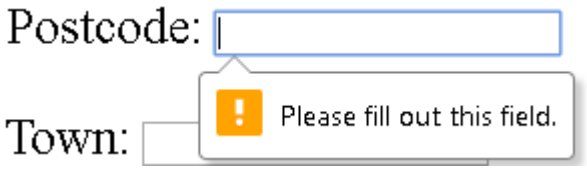
Result: pass

Edit Profile page



Test 29

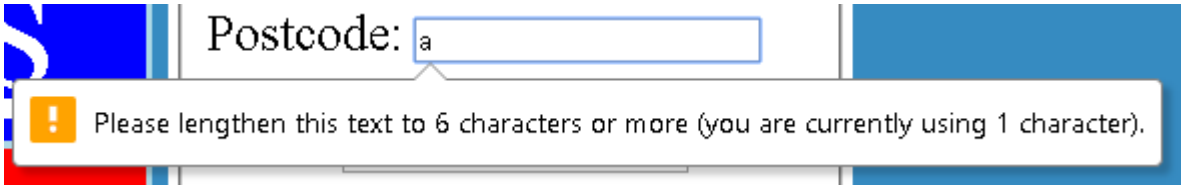
Submitting the form without filling in postcode will produce an error. Data type is typical. Expected result is to see an error



Result: pass

Test 30

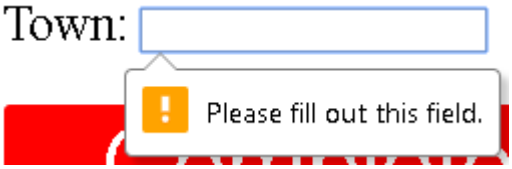
Postcode must be at least 6 characters long. Data type is erroneous. Expected result is to see an error.



Result: pass

Test 31

Submitting the form without filling in town will produce an error. Data type is typical. Expected result is to see an error.



Result: pass

Test 32

If banned word used in postcode an error is raised. Data type is erroneous. Expected result is to see an error. Postcode: fucked

Banned word was used. Please remove the word 'fuc' and try again

Result: pass

Test 33

If banned word used in town an error is raised. Data type is erroneous. Expected result is to see an error. Town: shit

Banned word was used. Please remove the word 'shit' and try again

Result: pass

Test 34

Entering valid information into the form will save the form data in the database and redirect the user back to their completed user information page.

Year: 13, Interest: Video games, postcode: CR78QG, town: Thornton heath

Interest table:

6	Video Games

Postcode table:

9	CR78QG

Town table:

7	thornton heath

Year Group table:

9	13

User information table:

12	6	9	7	12	9

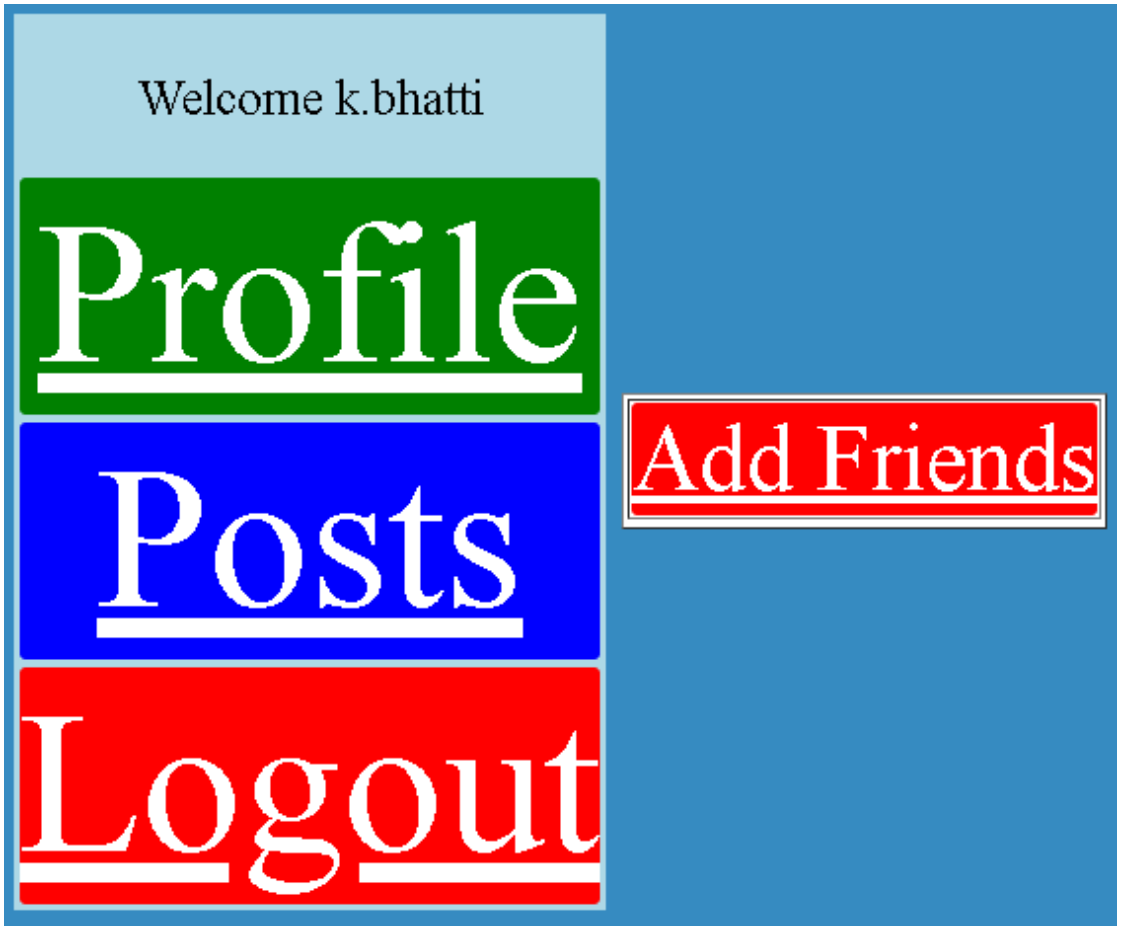
User table:

12	pbk...	201...	0	k.bhatti	kamran	k.bhatti@hfed...	0	1	201...	bhatti

Result: pass

Test 35

Clicking the friends' button takes the user to the friends' page. Data type is typical. Expected result is being able to view the friends' page.



Result: pass

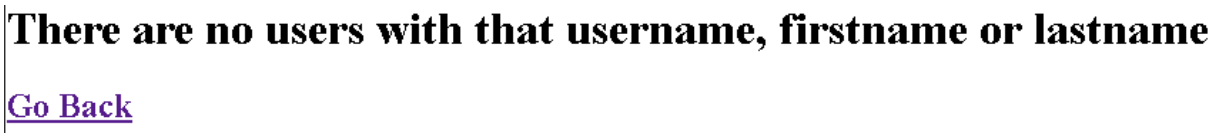
Test 36

Clicking the add friends button will take the user to the add friends page. Data type is typical. Expected result is the add friends page being displayed.

Result: pass

Test 37

If user clicks search without filling the search field or by filling the search field with data which doesn't relate to a users' username, first name or last name, an error page is displayed. Data type is erroneous. Expected result is to see an error page.



Result: pass

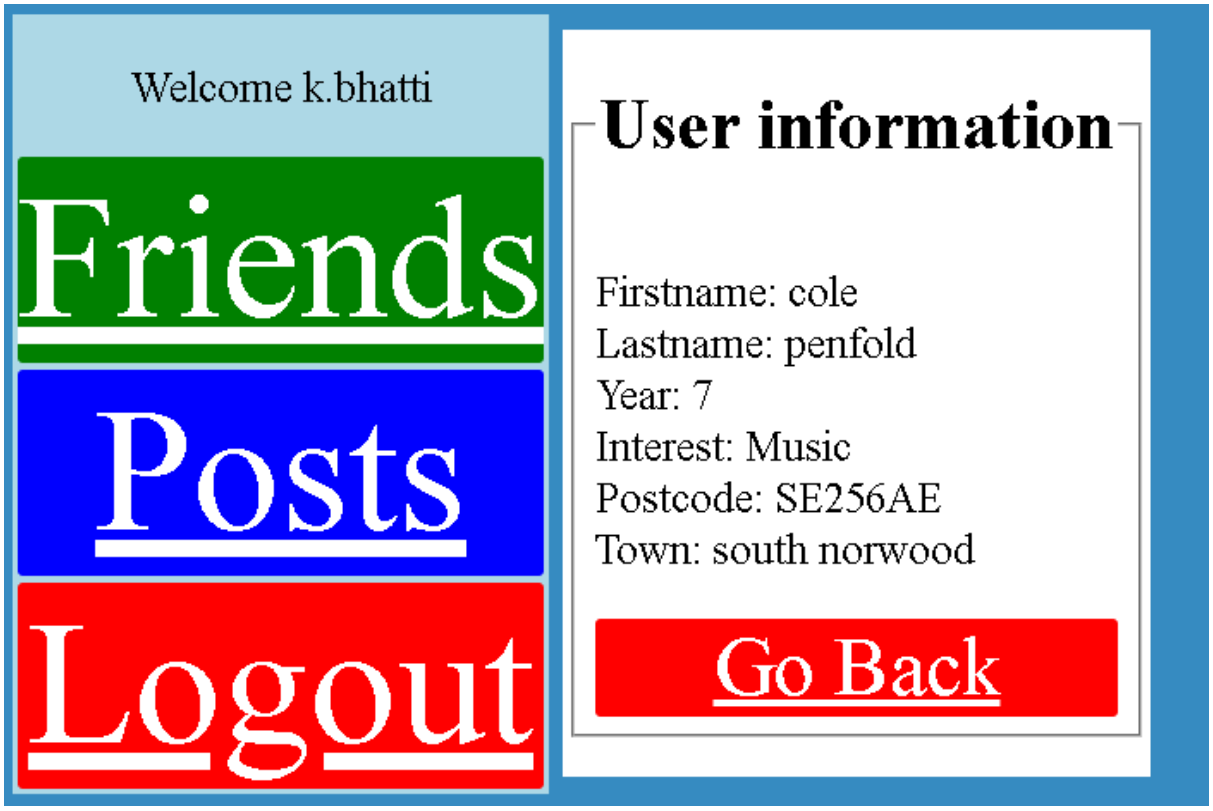
Test 38

Users are able to search for users not in their recommended lists. Data type is typical. Expected result is to see a user which is not in the recommended friends list.

Result: pass

Test 39

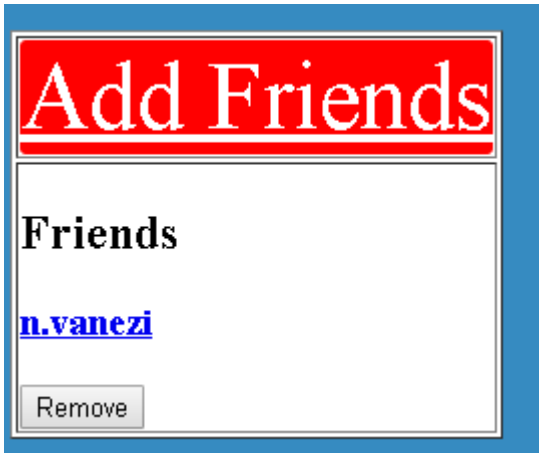
If user clicks on a users' username, their profile page is displayed. Data type is typical. Expected result is to see another users profile page.



Result: pass

Test 40

Clicking add on a user will add the user to the friends list of the current logged in user. Data type is typical. Expected result is for the other user to display in the current users friends list and for the relationship to save to the database and will redirect the user back to their friends' list page. Adding Nathan vanezi



In friend table

8	12

This shows that the current user is in the table friends thus the current user is the parameter for the many to many field.

In friend user table

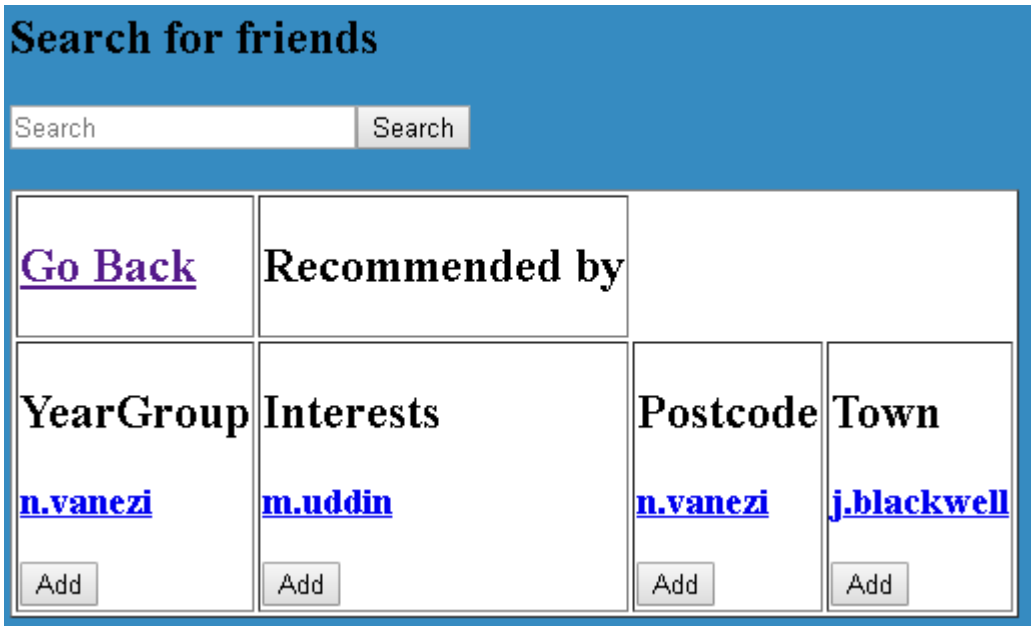
id	friend_id	user_id
Filter	Filter	Filter
12	8	19

This shows that the current user is friends with the other user.

Result: pass

Test 41

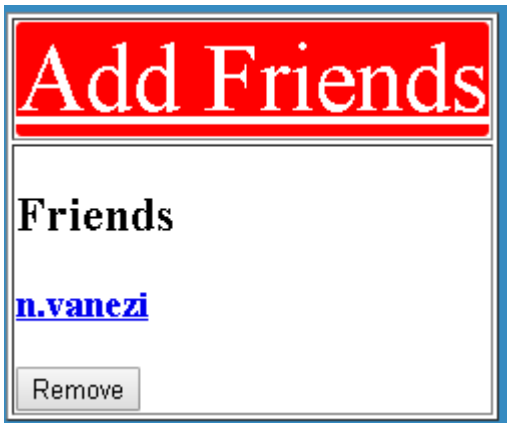
Looking for the added friend again will not show the add button. Data type is typical. Expected result is to not see an add friend button under n.vanezi in the add friends page.



Result: fail

Test 42

Clicking on the friend in the friends list to add will not add the friend to the friend list again. Data type is erroneous. Expected result is for the added user to not be added as a friend again.



Result: pass

Test 43

Clicking on the friend in the friends list will remove the friend from the current users' friends list and will delete the relationship from the table. Data type is typical. Expected result is for the friend to be removed from the friends list and the relationship between the users to be removed.

Friends table

8	12

This is unchanged as the record for the user in the friends table shows that the user can have friends.

Friends user table

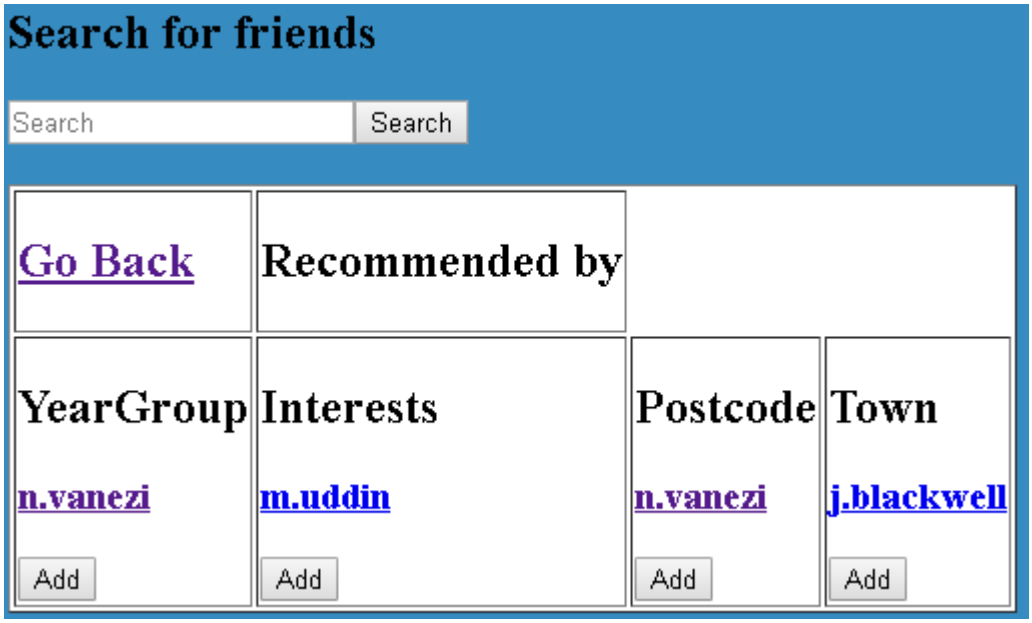
id	friend_id	user_id
Filter	Filter	Filter

This shows the relation with the other user has been removed.

Result: pass

Test 44

Clicking go back on a user's profile will take the logged in user to their recommended friends page. Data type is erroneous. Expected result is to be displayed the recommended friends page after clicking go back on a users' profile.



Result: pass

Test 45

Clicking go back on the friends search page or recommended friends page will redirect the user back to their friends' list page. Data type is erroneous. Expected result is to be redirected to the friends' list page.

Result: pass

Test 46[74]

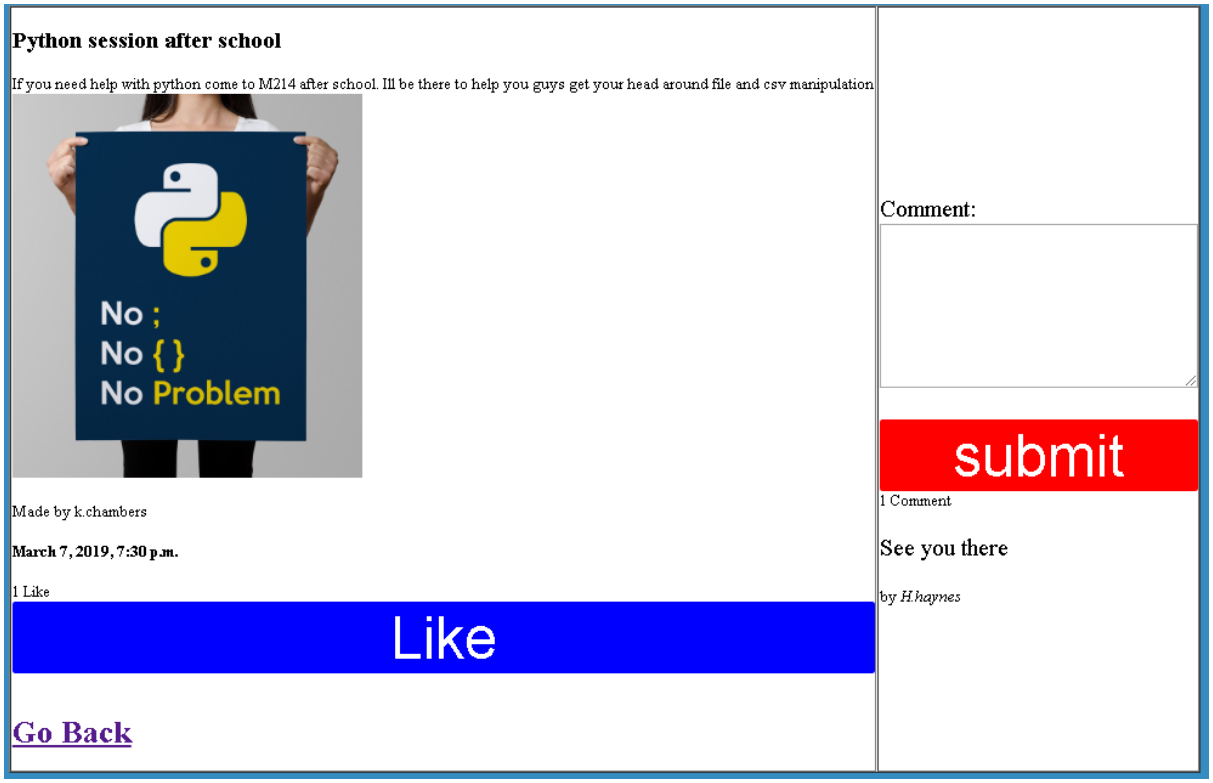
Clicking the post page will take a user to view a list of all the posts posted on the social network. Data type is typical. Expected result is to see a list of all the posts made by users and yourself.

[74] https://medium.com/@Intersog/some-cool-things-you-can-do-with-python-12a595164b49

Result: pass

Test 47

If user clicks on a post, it is displayed on its own with the user who created the post, title of the post, content of the post, image associated with the post if there is one, date the post was made, comments associated with the posts and number of likes associated with the post. Data type is erroneous. Expected result is to see the post in a separate page.



Result: pass

Test 48

User can like a post. Data type is typical. Expected result is for likes to increase and for their name to appear in the likes table associated to the post.

In posts likes table

13	13	12

The current user is 12 the post is 13

Result: pass

Test 49

User is able to unlike a post. Data type is typical. Expected result is for the like they just added to be removed and for their name to be disassociated from the post in the likes table.

In posts likes table

id	posts_id	user_id
Filter	Filter	Filter
11	13	15
12	12	17

Only one user has liked the post and the current user has been removed from liking the post.

Result: pass

Test 50

If comment has a banned word error is raised. Data type is erroneous. Expected result is for an error to be displayed.

Banned word was used. Please remove the word 'bollock' and try again

Result: pass

Test 51

If comment form is not filled on submit, an error is raised. Data type is typical. Expected result is for an error to be raised.

Result: pass

Test 52

User can add comment to a post. Data type is typical. Expected result is that if a valid comment is submitted, the comment is displayed on the page and the comment is saved in the database linked to the post.

2 Comments

I struggle on that soooo much

by *K.bhatti*

See you there

by *H.haynes*

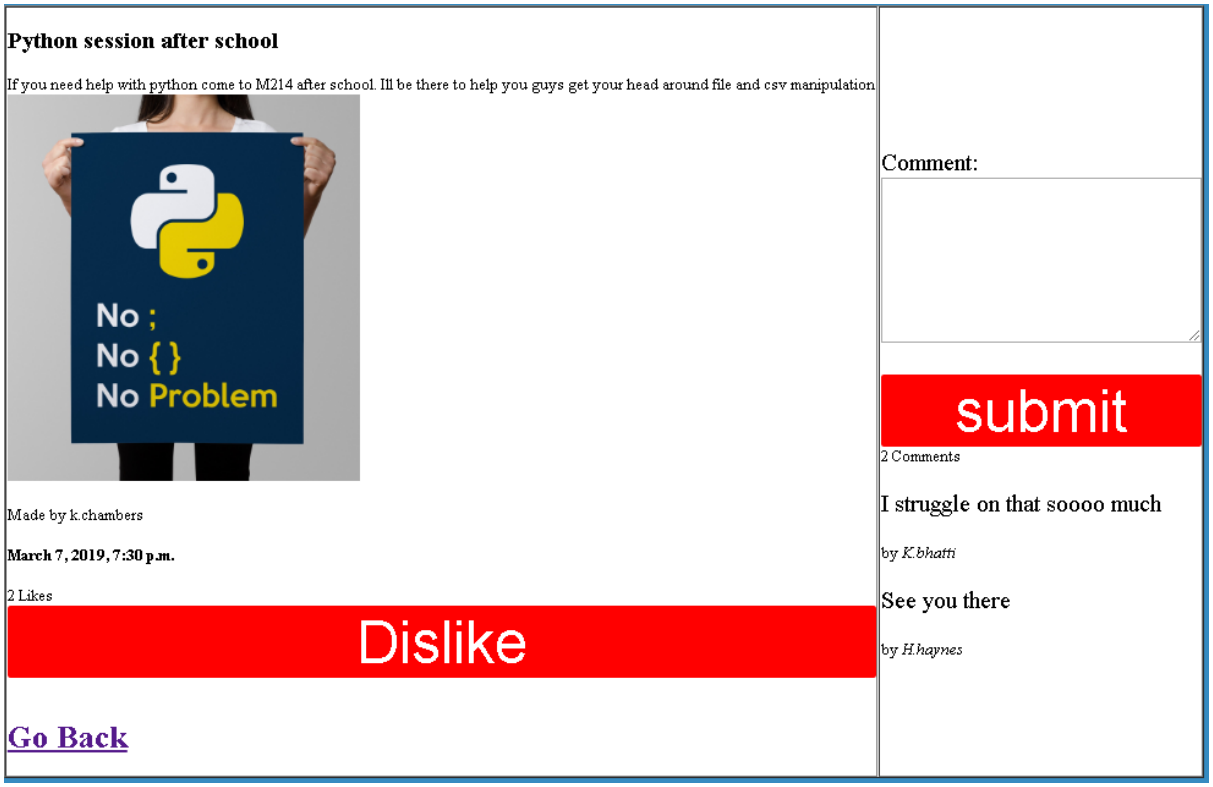
In comments table in database

14	I struggle on that soooo much	2019-03-07 1...	13	12

Result: pass

Test 53

If a user likes a post, they are still able to leave a comment. Data type is expected. Expected result is for a user to leave another comment after liking the post.



Result: fail

Test 54

If the user clicks go back after searching for a post on the post page or clicks go back after viewing a post in its separate page, the user is redirected to a list of all the posts created. Data type is typical. Expected result is to be redirected to a page which lists all the posts created.

Result: pass

Test 55

If user clicks create post, user is redirected to a form to create their own post. Data type is typical. Expected result is to be shown a form to create a post.

Result: pass

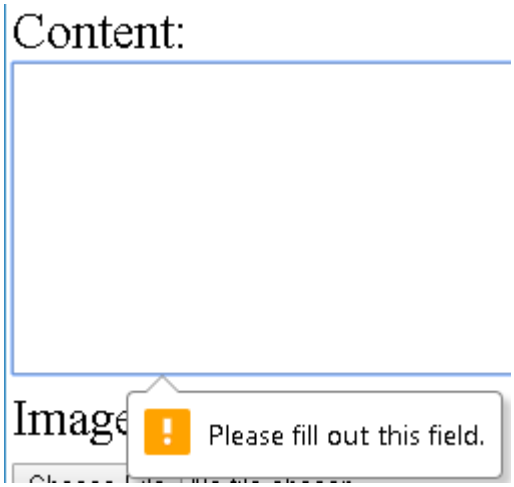
Test 56

If title is not filled when submitting the create post form an error is raised. Data type is typical. Expected result is to see an error.

Result: pass

Test 57

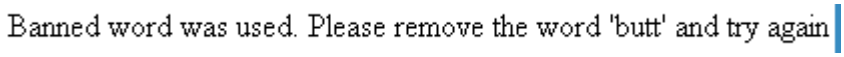
If content is not filled when submitting the create post form an error is raised. Data type is typical. Expected result is to see an error.



Result: pass

Test 58

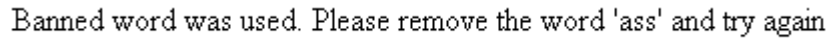
If banned word is used in title on submitting the create post form an error is raised. Data type is erroneous. Expected result it so see an error.



Result: pass

Test 59

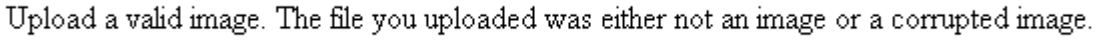
If banned words is used in content on submitting the create post form an error is raised. Data type is erroneous. Expected result is to see an error.



Result: pass

Test 59 part b

If file that is being uploaded is not an image file an error is raised. Data type is typical. Expected result is to see an error.



Result: pass

Test 60[75]

If form for create post is valid, the user is able to create a post with an image and the user is redirected to the post list page. Data type is typical. Expected result is for a post to be made with an image which is associated to the current logged in user.

[75] http://www.mrbartonmaths.com/blog/solving-equations-with-surds-gcse-maths-question-of-the-week-higher/

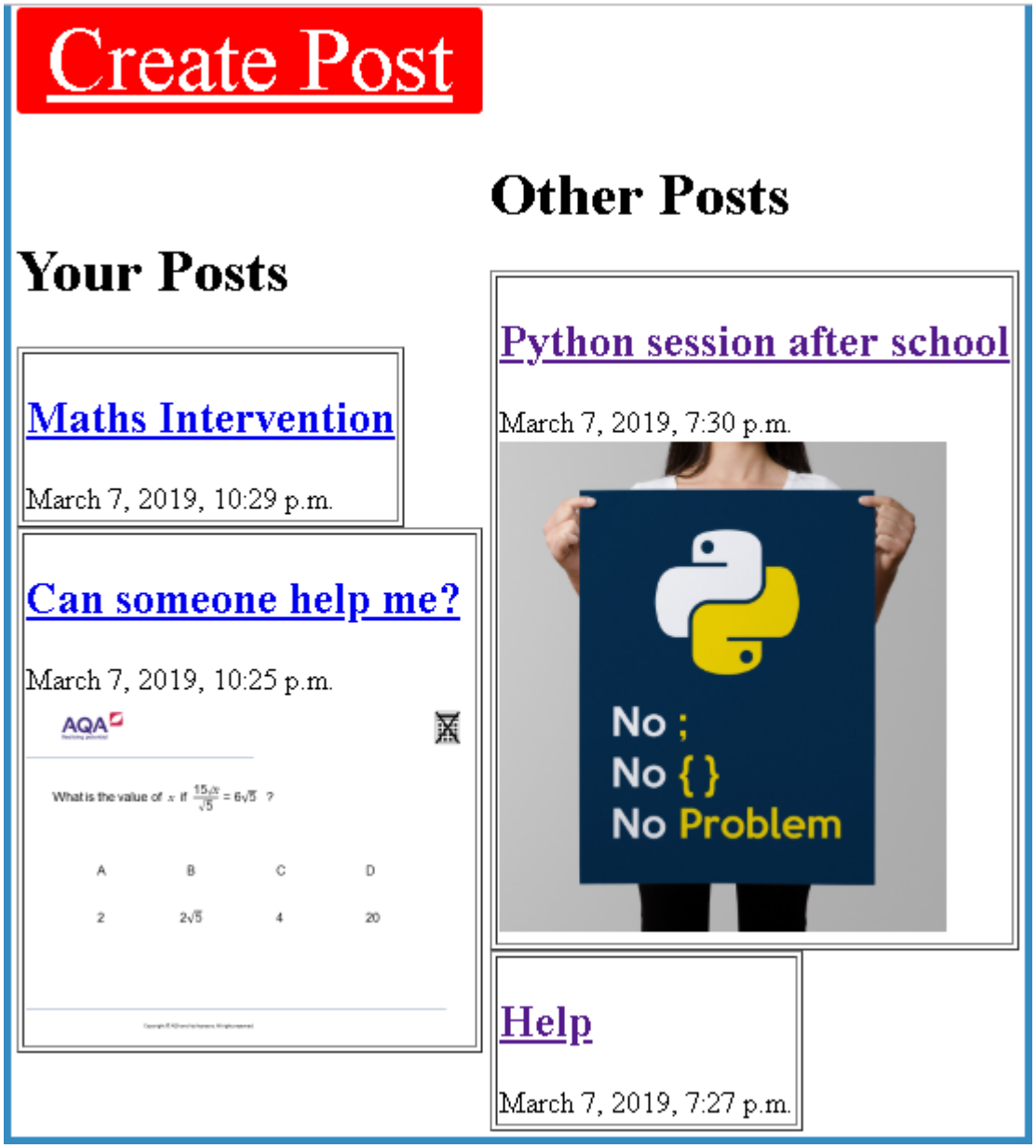
In posts table

14	Can someone help me?	Im stuck on a ...	post_images/...	12	2019-03-07 2...

Result: pass

Test 61

If form for create post is valid, the user is able to create a post without an image and the user is redirected to the post list page. Data type is typical. Expected result is for a post to be made without an image which is associated to the current logged in user.



In posts table

15	Maths Intervention	Begins next w...		12	2019-03-07 2...

Result: pass

Test 62

A user is able to delete their own post. Data type is typical. Expected result is for post to be deleted from the database and post list page.

In posts table

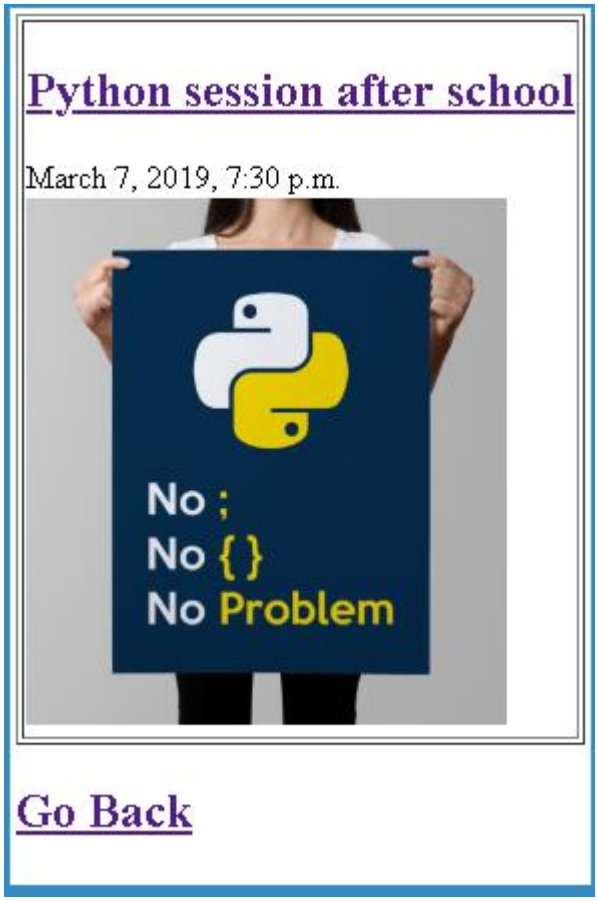
id	title	content	image	user_id	publish
Filter	Filter	Filter	Filter	Filter	Filter
12	Help	I need someo...		16	2019-03-07 1...
13	Python session after school	If you need h...	post_images/...	13	2019-03-07 1...
14	Can someone help me?	Im stuck on a ...	post_images/...	12	2019-03-07 2...

The post associated to the user has been deleted.

Result: pass

Test 63

Searching for a valid post title or content will display the post in the searched post page. Data type is typical. Expected result is to see all posts with the relevant query searched. Query = python



Result: pass

Test 64

If nothing is put in query all posts that have been created are displayed. Data type is erroneous. Expected result is for all the posts to be displayed.

Result: pass

Test 65

If query doesn't match any posts an error page is displayed. Data type is erroneous. Expected result is to see an error page.

Result: pass

Test 66

Clicking logout will log the user out of the social network and redirect the user to the index page. Data type is typical. Expected result is to see the index page after logging out.

Result: pass

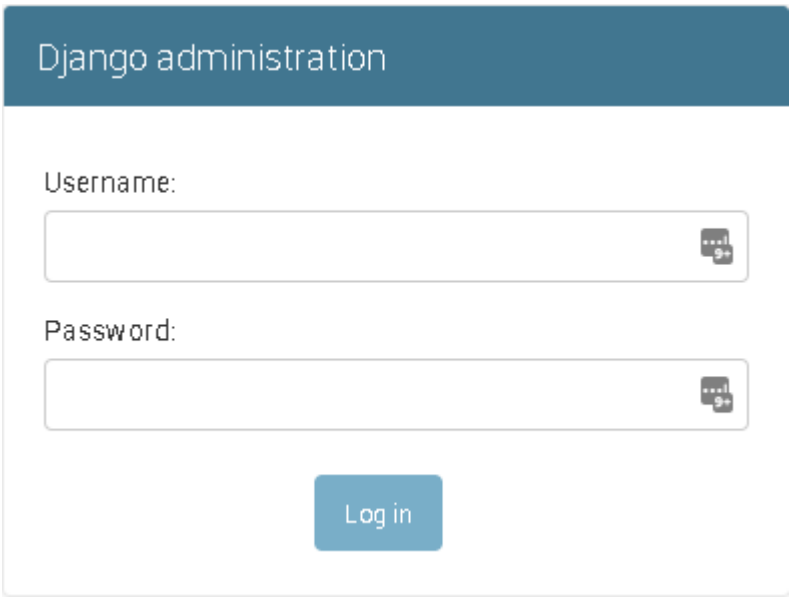
Test 67

Trying to access any of the pages after logging out without logging back in will display an error page. Data type is erroneous. Expected result is to see an error page.

Result: pass

Test 68

Clicking the admin button from the index page will take the user to the admin login page. Data type is typical. Expected result is to be redirected to the admin login page.



Result: pass

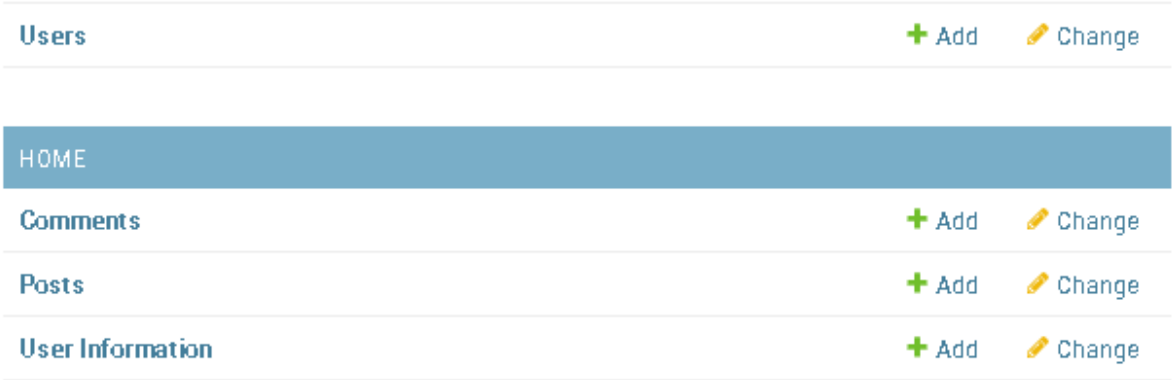
Test 69

Not entering the correct admin login details will display an error message on the admin login page. Data type is typical. Expected result is to see an error message.

Result: pass

Test 70

After logging in the admin is able to see tables in the database. Data type is erroneous. Expected result is a view of specific tables in the database.



Result: pass

Test 71

Clicking the comments, posts, and user information and users links will display their respective table. Data type is typical. Expected result is to see all the data in the tables in a readable format.

User table

USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
Kamran	kamranbhatti03072001@gmail.com			✓
c.penfold	c.penfold@hfed.net	cole	penfold	✗
h.haynes	h.haynes@hfed.net	henry	haynes	✗
j.blackwell	j.blackwell@hfed.net	Joseph	Blackwell	✗
k.bhatti	k.bhatti@hfed.net	kamran	bhatti	✗
k.chambers	k.chambers@hfed.net	kehron	chambers	✗
m.uddin	m.uddin@hfed.net	masum	uddin	✗
n.vanezi	n.vanezi@hfed.net	Nathan	Vanezi	✗
t.duncan	t.duncan@hfed.net	tyrik	duncan	✗

User information table

USER	YEAR	INTEREST	POSTCODE	TOWN
n.vanezi	13	Reading	CR78QG	merton
t.duncan	8	Reading	SE278AF	south norwood
h.haynes	9	Tv or Movies	SE248PF	south norwood
j.blackwell	12	Music	CR8JEP	thornton heath
m.uddin	10	Video Games	CR06BE	croydon
c.penfold	7	Music	SE256AE	south norwood
k.chambers	11	Art	SEN46E	south norwood
k.bhatti	13	Video Games	CR78QG	thornton heath
Kamran	-	-	-	-

Posts table

USER	TITLE	PUBLISH
k.bhatti	Can someone help me?	March 7, 2019, 10:25 p.m.
k.chambers	Python session after school	March 7, 2019, 7:30 p.m.
j.blackwell	Help	March 7, 2019, 7:27 p.m.

Comments table

POST	USER	PUBLISH
Python session after school	k.bhatti	March 7, 2019, 7:56 p.m.
Python session after school	h.haynes	March 7, 2019, 7:39 p.m.
Help	m.uddin	March 7, 2019, 7:39 p.m.

Result: pass

Test 72

Admin can search in user, posts, comments and user information table. Data type is typical. Expected result is to see the records after making a query to each table. Query is kamran

User table

USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
Kamran	kamranbhatti03072001@gmail.com			
k.bhatti	k.bhatti@hfed.net	kamran	bhatti	

Posts table

USER	TITLE	PUBLISH
k.bhatti	Can someone help me?	March 7, 2019, 10:25 p.m.

Comments table

POST	USER	PUBLISH
Python session after school	k.bhatti	March 7, 2019, 7:56 p.m.

User Information table

USER	YEAR	INTEREST	POSTCODE	TOWN
k.bhatti	13	Video Games	CR78QG	thornton heath
Kamran	-	-	-	-

Result: pass

Test 72

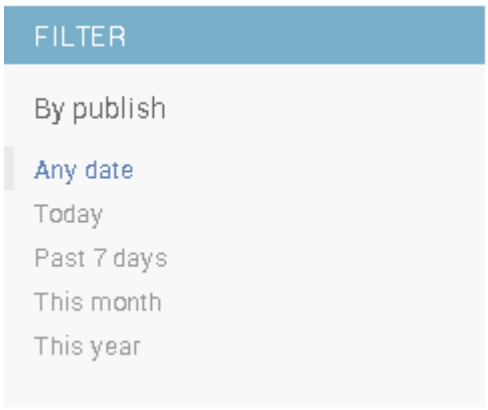
Admin can search in user, posts, comments and user information table. Data type is typical. Expected result is to see no records after making a query to each table. Query is x

0 User Information 0 users 0 comments 0 Posts

Result: pass

Test 73

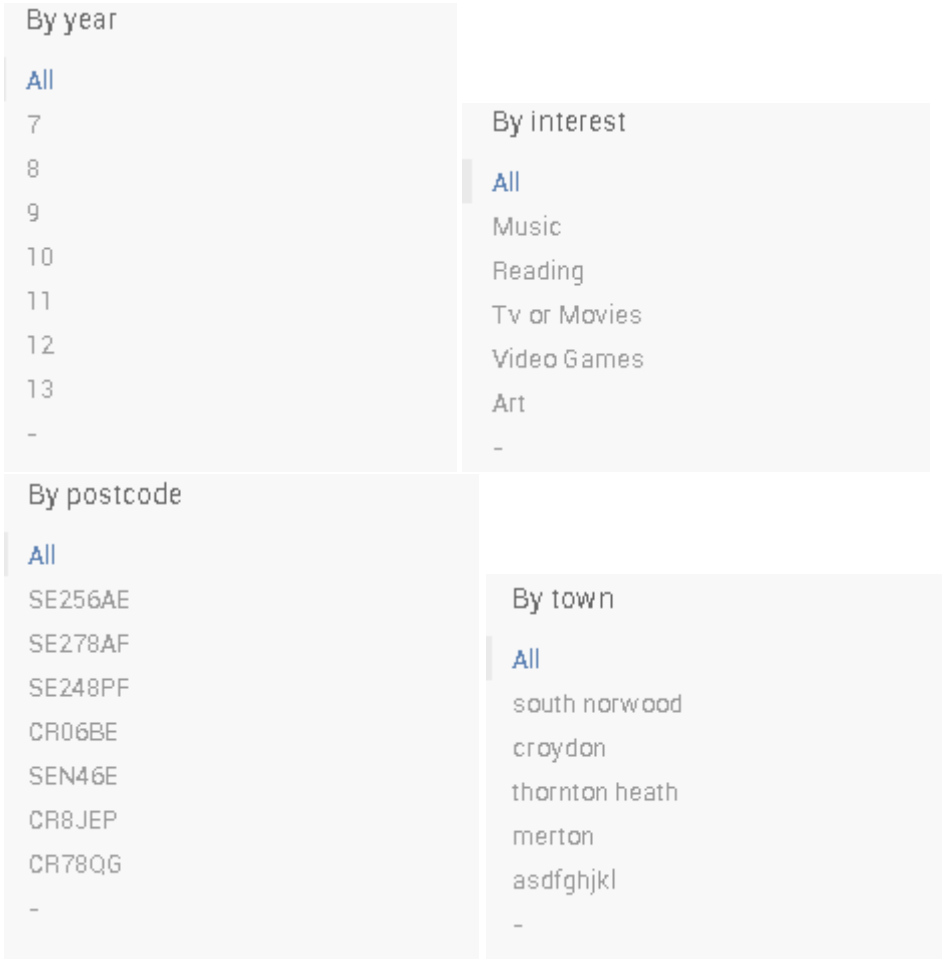
Admin is able to filter by date published on posts and comments table. Data type is erroneous. Expected result is to see a filter for publish date on side of table for posts and comments.



Result: pass

Test 74

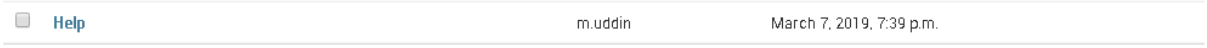
Admin is able to filter by year, interest, postcode and town. Data type is erroneous. Expected result is to see a filter for year interest, postcode and town on side of table.



Result: pass

Test 75

If comment is deleted from admin page then comment is deleted from database. Data type is typical. Expected result is for comment to be removed from database. Removing:



After removing:

In comment table

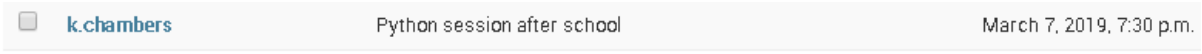
id	content	publish	post_id	user_id
Filter	Filter	Filter	Filter	Filter
13	See you there	2019-03-07 1...	13	17
14	I struggle on that soooo much	2019-03-07 1...	13	12

Comment was removed.

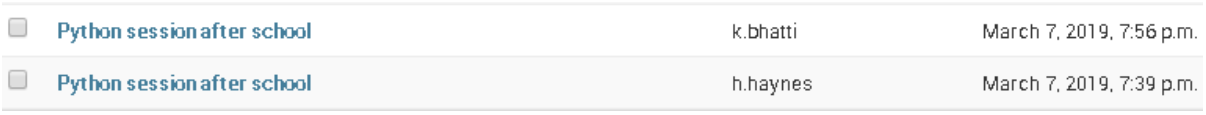
Result: pass

Test 76

If post is deleted from admin page then comment and post is deleted from database. Data type is typical. Expected result is for comment and post to be deleted from database. Post to remove:



Comments linked to post:



After removing:

In posts table

id	title	content	image	user_id	publish
Filter	Filter	Filter	Filter	Filter	Filter
12	Help	I need someo...		16	2019-03-07 1...
14	Can someone help me?	Im stuck on a ...	post_images/...	12	2019-03-07 2...

In comments table

id	content	publish	post_id	user_id
Filter	Filter	Filter	Filter	Filter

Post and comments linked to post were removed.

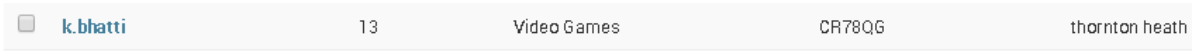
Result: pass

Test 77

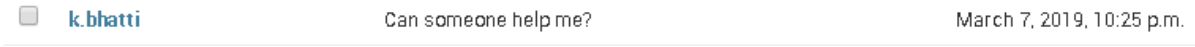
If user is deleted, user information and posts for that user are deleted. Data type is typical. Expected result is for user, user in user information and users posts to be deleted.

Removing:

With user information:



And post:



After removing:

In user table

id	ısswoı	ɜt_log	uperu	username	first_name	email	ɜ_staf	_activ	e_joir	last_name
Filter				Filter	Filter	Filter				Filter
1	pbk...	201...	1	Kamran		kamranbhatti03072001@gmail.com	1	1	201...	
13	pbk...	201...	0	k.chambers	kehron	k.chambers@hfed.net	0	1	201...	chambers
14	pbk...	201...	0	c.penfold	cole	c.penfold@hfed.net	0	1	201...	penfold
15	pbk...	201...	0	m.uddin	masum	m.uddin@hfed.net	0	1	201...	uddin
16	pbk...	201...	0	j.blackwell	Joseph	j.blackwell@hfed.net	0	1	201...	Blackwell
17	pbk...	201...	0	h.haynes	henry	h.haynes@hfed.net	0	1	201...	haynes
18	pbk...	201...	0	t.duncan	tyrik	t.duncan@hfed.net	0	1	201...	duncan
19	pbk...	201...	0	n.vanezi	Nathan	n.vanezi@hfed.net	0	1	201...	Vanezi

In user information table

id	interest_id	postcode_id	town_id	user_id	year_id
Filter	Filter	Filter	Filter	Filter	Filter
1	*NULL*	*NULL*	*NULL*	1	*NULL*
13	7	7	5	13	7
14	3	3	5	14	3
15	6	6	6	15	6
16	3	8	7	16	8
17	5	5	5	17	5
18	4	4	5	18	4
19	4	9	8	19	9

In Posts table

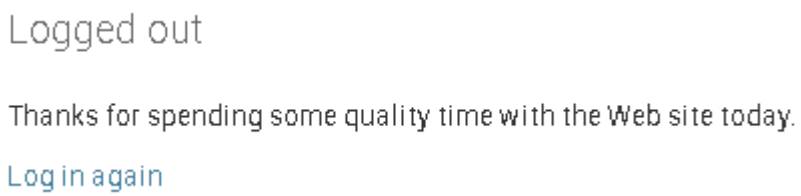
id	title	content	image	user_id	publish
Filter	Filter	Filter	Filter	Filter	Filter
12	Help	I need someo...		16	2019-03-07 1...

All fields or records in database associated with user are removed.

Result: pass

Test 78

Admin is able to log out of admin page and be redirected to a logout page. Data type is typical. Expected result is for admin to have a logout page.



Result: pass

Test 79

Ability to block or report users as a student. Data type is typical. Expected result is to see a form after logging in or viewing a post or comment or user profile and being redirected to a form to report users for misconduct on the social network.

Result: fail

EVALUATION

Have I met my objectives?

Number	Objective	Met
1	Display index page to redirect to admin or student login page when clicked	Yes
2	Validate admin and student login details then redirect to their respective pages (admin to admin page, student to profile page) or see form errors.	Yes
3	Redirect to signup page when selected, allow users to sign up, and validate signup details by loading a file with banned words and redirect to login page after signing up if details are appropriate or see form errors	Yes
4	Redirect to all student pages from the profile, posts and friends pages	Yes
5	Allow users to edit their profile and validate details entered on the user information page by loading a file with banned words and store details in database if details are appropriate or see form errors	Yes
6	Allow students to create posts then validate created post by loading a file with banned words to see if the content in the post has inappropriate words then store post details in database if post details are appropriate or show form errors	Yes
7	Allow students to comment on posts and validate comments on a students' post by loading a file with banned words to see if the content in the comment has inappropriate words. Store comment details in database is comment is appropriate	Yes
8	Display other posts that students can see	Yes
9	Allow students to delete posts from database if they, the owner of the post selects it to be deleted and ensure users are not able to delete other users' posts.	Yes
10	Create relationship between users in database if they are friends or remove relationship between users in database if they cease to be friends when a student adds, removes or searches for a friend. Additionally ensure students are not able to delete the friends of other users.	Yes
11	Validate pages accessed by students so that unauthorised users aren't able to access the social network	Yes
12	Client server model to access social media network from anywhere with an internet connection.	Yes
13	student can view their profile to edit their own profile and are able to view other users profiles yet aren't able to edit other users profiles	Yes
14	Users can upload image with posts	Yes
15	allow student to search for posts	Yes
16	Admin has the ability to delete student accounts, student posts or comments on students posts	Yes
17	Students can like or dislike posts	Yes
18	Students have the ability to block or report individuals as a student	No

Display the index page and redireting to the admin login and student login didn't take much time as I already had knowledge on how to implement this in HTML so the transition to use a url structure based in python and referencing the urls in the HTML document was not difficult to understand or carry out. Therefore, further implementation of redirecting to other pages was not difficult to implement as I had a predefined structure of how to implement buttons and redirects on form submissions due to the index page and student login page.

Allowing users to sign up to the systemwas difficult to understand as I initially tried to make my own user table to act as the login system and I tried to store all the profile information as the user was creating an account for the system. Once I learnt how to implement the built in User table in django creating accounts and redirecting users to the login page was much more simplistic than the way I attempted to implement it initially.

Implementing the database structure for the user information table was very difficult as I could not make direct connections to the user table to store the foreign keys for the year, interest, postcode or town and instead I implemented the database structure after understanding that the data being called by each foreign key in another table wasn't the actual data in the field in the other table but the primary key of the record in the other table. This made implementation of relational structures like the posts and comments tables much easier to understand due to initially having to implement the complex strucutre of the user information table.

Displaying other users posts was quite simple to implement as I would just have to get all the users primary keys and display each users associated post with that posts title, image and the date the post was published which I could implement simply in a for loop. This made implementing the friends list very simple as i would have to user the exclude and filter parameter in both scenarios to display the current users posts and other users posts seperatley which is ver similar to display all the users with the same year group or postcode whilst excluding the currenly logged in user.

Allowing users to delete their own posts was very simple as it only required me to program the object in the database being deleted. The difficulty in implementing this however is that redirecting the user so that after the post was deleted, the user would be directed to the posts page was difficult to implement as I had to use functions such as HttpResponseRedirect in a different was in means of returning to a previous page instead of increasing the length of the url by going to a page associated with the currently viewed page. However, after implementing this feature implementing features to remove friends or add likes to a post allowed me to understand the concepts behind its implementation better thus allowing me to implement these features in a form simpler to what I had anticipated in designing the project.

Creating relationships between users was difficult for me to understand at first due to me not understanding how to implement this funcitonalty even with an unnormalised database. However, upon manipulating the posts for users and profiles for users, I was able to understand that I needed to create a function in means of implementing the add and remove friends functionality correctly as I was using a many to many relational structure between users and friends of users in the database table for friends.

The client server model was not difficult to implement as the files required to run the server are pre-defined and autogenerated by django so it was not necessary for me to edit how the server was implemented as for my experience, the way in which the server was run was sufficient for my usage in testing and development.

As I had previously manipulated primary keys for the posts table and when implementing the relationships for the user information table, the use of primary keys to specify which users' profile imformation to receive from the database was not difficult when implementing functionality for students being able to view other students profiles.

Implementing functionality for users to upload images with posts was difficult for me to understand. This is due to the documentation for django not being bery clear in where the static path should be located in urls.py as there were multiple urls.py in my project and the documentation didn't specify

the type of HTML forms needed to receive files through a form. However, after using multiple sources to understand that I has been using the wrong HTML formset and by using trial and error to position the static path in urls.py correctly I was able to implement uploading images with posts correctly.

Allowing for students to search for posts and friends was difficult for me to understand as initially I had tried to use the filtering method on the fields in the database tables to locate specific records however, afer reading the documentation for django and understanding that I had to use the Q module when querying the database from a form filled by an end user, implementation of this search feature was quite straightforward.

In contrast, I believe the simplist aspect of the project was implementation of administrative access to the database tables as the only code I had to implement was the tables which the administrator could have access to and how the table can be queried as the interface for the tables in the administrative panel for django was automatically created.

Furthermore, implementation of the banned words text file was quite simplistic in execution as I am accustomed to file manipulation from the Pre Release material for computer science thus implementing this feature for text fields in multiple forms throughout my system was not difficult.

However, the most difficult aspect of the project would be the implementation of likes. I believe this difficulty was foreshadowed in my design in which I created a separate table for the likes and dislikes a post would have to which I finally implemented this functionallity as a many to many field in the post table to count the number of students which like a post.

Unfortunately, I was not able to implement the blocking or reporting feature within the project due to time constraints. I found this confusing to implement as I understood that the reporting feature could be displayed as a HTML form to be displayed to the user in which they could select a username to report or block but I was unsure how the administrator would be able to view this form data. If the data was submitted as a form then I could have implemented the reporting or blocking feature as either a separate one to many relationship between the reported user and the many users which have reported the user or as a boolean field in the user table which could restrict access to the site for that user if it is set to true. Although I understand how to implement the student half of the reporting system I find it difficult to grasp how the administrator would interface with the system in means of either removing the users access to the account or suspending the users access to the account till their accounts' data is reviewed.

Feedback

How do you feel about the overall design of the project?

"You have ensured that a specific theme is eminating from your project as the colour scheme is similar in most pages where you have darker colours for pages that are accesible to users who arent logged into the system and then the contrast with the lighter colours after logging in seems refreshing in its design as it doesn't seem as harsh as the darker colours on the index, student login and sign up pages. The wide span use of roll over buttons shows that you have taken careful consideration in how you would like certain aspects of the application to be customised to entice users to use the system.

However, having different themes for the index, signup and login pages with the contrast to the pages after signing in is very strenuating and difficult to adjust to when at night. Considering that the target audience for the software is students which tend to stay up late, implementing theme which reduces eye strain may entice users to use this network over other forms of social media like facebook which doesn't have this feature implemented. Furthermore, the form fields are quite small on wide screen monitors which is very difficult to read whereas if you implemented dynamic form fields the change in aspect ratio would not be a major affector of the overall usability of the project."

What do you believe was implemented well in the project?

"The project seems very suitable for a school setting as there will be minimal exposure of students to indecent or profane content due to the algorithm you have used to filter profane content from text fields on the network. Furthermore, you have paid significant detail to the design of recommending friends to users in ensuring that only fields with friends inside them show and by ensuring that users can clearly see their recommendation of friends by year, interest, town and postcode. Also, the implementation of querying for friends and posts is very unique in which you have a specified error page for no results being returned and the error page when trying to access any links on the network without logging in coincides with the general theme that you are trying to portray. Finally, the implementation of filters and search fields that you have specified for the tables on the administrative page is very useful when trying to locate a specific user, post or comment."

What features do you think I have not implemented well?

"The interest field seems restrictive in the amount of interests you select which may be offputting to users as they arent able to be more specific or express themselves greatly with their profiles whilst on networks like facebook, this isnt an issue. Although you have implemented a feature to recommend friends, having multiple lists of recommended friends being displayed to a user seems overwhelming and having a list with a select few friends seems more reasonable as you have already implemented functionality to search for friends to add.

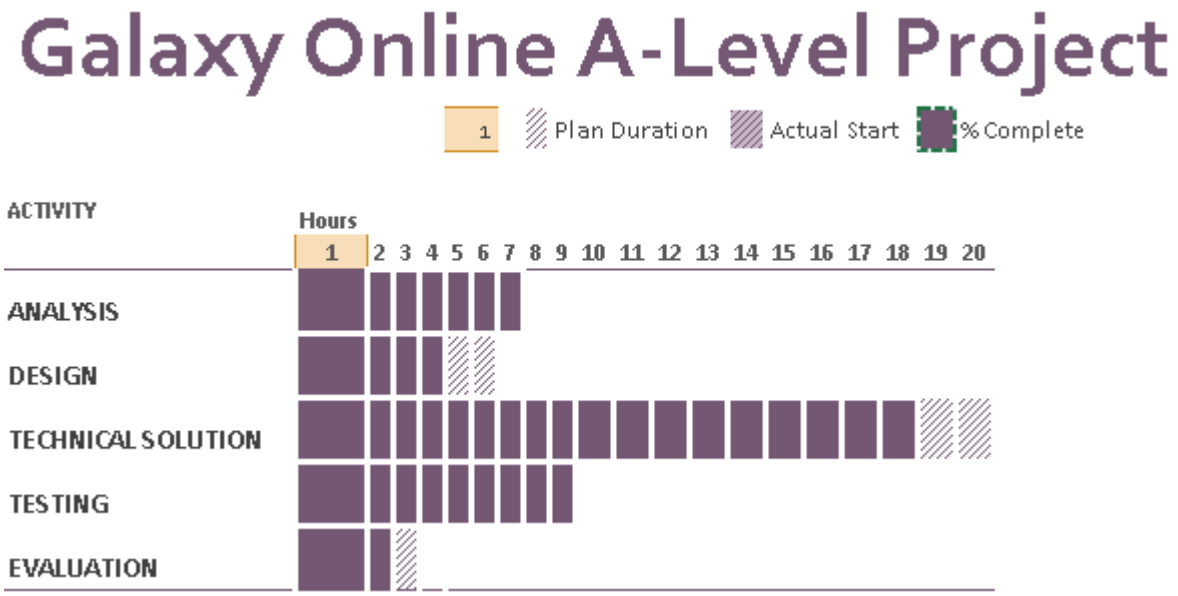
I believe the main issue in the project would be the implementation of likes and comments as although the user is able to like or comment, the user is unable to do both which may deter users from using your proposed system. Finally, if the search fields for posts and friends were given better customisation such as custom rollover buttons or being placed in tables to improve the overall design of the search fields to make them more visible or prominent the system would seem more consistent in its themeing."

Conclusions from feedback

I feel as though I focused too heavily on making the system look presentable to an end user to which I lost sight of the main reason why I intended to make the system which led to faults such as the like and comment system not working correctly when both are implemented or the search fields not having customised buttons whilst all the other buttons are customised. I believe that these details and the contrast with the login, index and sign up pages will be the main reasons end users will not favour my system.

If I were to attempt to rectify the project or attempt to create a similar project I would initially identify and consider implementing data structures and main features of the system before focusing on the customisation of the webpage and ensure that I understand completely how I would implement a data structure into my courswork such as a database structure. This is so that I wont have issues with normalisation of the database or issues when trying to implement the individual tables and relationships in the database so that I can spend more time on implementing key algorithms unlike my approach to implement the database in this project while not having a full understanding of how I will be manipulating data.

Gannt Chart



I believe I have used my time efficiently in means of completing the coursework within technical solution, testing and evaluation and design however, I feel that I spent too much time in analysis which has caused an oversight in my implementation. The reporting or blocking feature has not been implemented, the friends' relationship has been implemented as a many-to-many relationship in the database and the comments and likes system is not working coherently as a user is only able to comment or like not do both. I believe if I had spent less time working on the analysis for the project, the end user would be more satisfied with using the system as I would be able to fix these errors by redesigning the relational database to incorporate these features correctly.

Extensions I could make

For the initial forms on the website I could use Djangos inbuilt crispy forms so that the forms resize and are able to be easily viewable on a widescreen monitor to reduce the eye strain of users who are using the project at different times in the day. Furthermore, I could use bootstrap for customising the buttons on the website and the layout of forms on the screen so that I can spend more time programming instead of customising the look of the product the end user will be interacting with. The use of the crispy forms and bootstrap will also make the form and field errors more readable and more distinguishable thus allowing users to notice errors in forms by the highlighted field and pop up boxes around the field instead of the field errors appearing near the area to submit the form. The use of bootstrap will also allow the form and body elements to be customised in further detail.

Additionally, if I replaced the drop down field on interests with a text input field, it would allow users to enter in their specific interest which would mean a greater range and more personalised interest to the users. This may intise users to use the system in a school setting in means of sharing their interests with their peers easily. Furthermore, if I had a database or text file with each postcode and its designated town, it would reduce the amount of validation I would need on the postcode and town fields. This would reduce the amount of malicous activities which may occur on the social network as a user with the intent to be malicious in stating that their postcode and town are valid when they are not would be detected.

Also, if I were to store the images uploaded to the project as a url thus hosting the image on an image hosting service such as flickr, I could retrieve the image as a url and save the url into the database. By utilising this I can access the post with sql queries to retrieve the image associated with the url while accessing or displaying its relevant associated fields being the title of the post, content in the post, who the post belongs to and when the post was created. This would reduce the amount of physical space being stored on the server or host computer as with the current implementation the file is uploaded and saved onto the storage device of the system (the hard drive) in a specified file directory whereas refering to an image hosted on a site like flickr or pinterest would only take up the storage necessary to store the url to the image which is significantly smaller in comparison to storing the image locally.

Lastly, if the friends list was represented using a graph algorithmand as one list then the user would be less confused and wont feel bombarded with information whenever the user wants to add friends. I could implement a single recommended friends list which would consist of having users recommended due to the users sharing the same year group, interest, postcode or town and have the users for each filter stored in a list. I could then filter the lists to only store unique values in all the lists and display the friends in the unique list to add on the add friends page of the social network. The graph algorithm could be used to recommend friends associated with a friend in the current users friends list in which the recommended friends lists will update due to specific filters like only recommending friends which have a common interest in physical activities likes sports. Friends can also be recommended due to the length or type of content they post so if the majority of their posts consist of a specific topic like maths, the current user is recommended friends due to the majority of the recommeneded users having posts with a similar theme.

I would also choose to deploy the application on a server so that users will be able to access the application without having to use my computer as a server to host a connection for users to interact with the project. This would allow a wider vareity of users to interact with the project without being limited by the connection that my computer can provide while running the server. Furthermore, if the admin was able to add words to the banned words file via the admin page, the system would be

able to catch more errors thus users of the system would be exposed to less profane content. This could be implemented by reading the file into the admin interface or a database table that would compare if values inputted in forms are in the data structure that stores the banned words. The admin could add records to the table from the admin page to expand the range of which profane content is detected thus reducing the exposure to profane content that a student would experience.

Bibliography

1. Zimmerman, A. (2012). *Online Aggression : The Influences of Anonymity and Social Modeling*. [online] Digitalcommons.unf.edu. Available at: https://digitalcommons.unf.edu/cgi/viewcontent.cgi?article=1472&context=etd
2. The Daily Cougar. (2016). *Social media encourages antisocial, aggressive behavior*. [online] Available at: http://thedailycougar.com/2016/07/27/social-media-encourages-antisocial-aggressive-behavior/
3. Bernard John, K. and Patience Emefa, D. (2018). *Effect of Social Media on Academic Performance of Students in Ghanaian Universities: A Case Study of University of Ghana, Legon*. [online] Digitalcommons.unl.edu. Available at: https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=4687&context=libphilprac
4. Asemah, E., Okpanachi, R. and Edegoh, L. (2013). *Influence of Social Media on the Academic Performance of the Undergraduate Students of Kogi State University, Anyigba, Nigeria*. [online] Pdfs.semanticscholar.org. Available at: https://pdfs.semanticscholar.org/2b7f/1561d3fed7fe48ec5c88281f7d7d9b74e70d.pdf
5. Apuke, O. (2016). *The Influence of Social Media on Academic Performance of Undergraduate Students of Taraba State University, Jalingo, Nigeria*. [online] Research Gate. Available at: https://www.researchgate.net/publication/317083347_The_Influence_of_Social_Media_on_Academic_Performance_of_Undergraduate_Students_of_Taraba_State_University_Jalingo_Nigeria
6. Schurgin O'Keeffe, G. and Clarke-Pearson, K. (2011). *Clinical Report—The Impact of Social Media on Children, Adolescents, and Families*. [online] Landspitali.is. Available at: https://www.landspitali.is/library/Sameiginlegar-skrar/Gagnasafn/Klinisk-svid-og-deildir/Kvenna--og-barnasvid/Gogn-fyrir-radstefnur/Hinn-gullni-medalvegur-Fyrirlestrar-2017/Clinical-Report_The-Impact-of-Social-Media-on-Children-Adolescents-and-Families.pdf
7. Chalk, A., Brennan, S. and Reed, M. (2018). *Safety Net: Cyberbullying's impact on young people's mental health Inquiry report*. [online] Childrenssociety.org.uk. Available at: https://www.childrenssociety.org.uk/sites/default/files/social-media-cyberbullying-inquiry-full-report_0.pdf
8. Acog.org. (2016). *Concerns Regarding Social Media and Health Issues in Adolescents and Young Adults - ACOG*. [online] Available at: https://www.acog.org/Clinical-Guidance-and-Publications/Committee-Opinions/Committee-on-Adolescent-Health-Care/Concerns-Regarding-Social-Media-and-Health-Issues-in-Adolescents-and-Young-Adults
9. Gdhr.wa.gov.au. (2019). *Social media: Cyberbullying - View - GDHR Portal*. [online] Available at: https://gdhr.wa.gov.au/-/cyberworld-cyber-bullying
10. Static1.squarespace.com. (n.d.). *Social Media Dangers and its Impact on Cyberbullying*. [online] Available at: https://static1.squarespace.com/static/55aeb103e4b03ff1478dc09f/t/58b6f382e6f2e16000428b9f/1488384903033/CyberSafetyNewsletterMMAFINAL.pdf
11. the Guardian. (2017). *Is social media bad for young people's mental health?*. [online] Available at: https://www.theguardian.com/mental-health-research-matters/2017/jan/20/is-social-media-bad-for-young-peoples-mental-health
12. Bullying.co.uk. (n.d.). *Effects of cyberbullying - Family Lives*. [online] Available at: https://www.bullying.co.uk/cyberbullying/effects-of-cyberbullying/
13. Garett, R., Lord, L. and Young, S. (2016). *Associations between social media and cyberbullying: a review of the literature*. [online] U.S. National Library of Medicine. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5344141/
14. Kelly Wallace, C. (2015). *Sexting: Majority of teens admit doing it - CNN*. [online] CNN. Available at: https://edition.cnn.com/2014/11/18/living/teens-sexting-what-parents-can-do/index.html
15. Ringrose, J., Gill, R., Livingstone, S. and Harvey, L. (2012). *A qualitative study of children, young people and 'sexting' A report prepared for the NSPCC*. [online] Lse.ac.uk. Available at: http://www.lse.ac.uk/media@lse/documents/MPP/Sexting-Report-NSPCC.pdf

16. Bajaj, R. (2017). *Top 10 Most Popular Social Sites and Apps in 2017*. [online] Linkedin.com. Available at: https://www.linkedin.com/pulse/top-10-most-popular-social-networking-sites-apps-2017-rajiv-bajaj
17. Bourgeois, D. (2018). *How to Set Up a Realtor Facebook Page in 9 Steps*. [online] Fit Small Business. Available at: https://fitsmallbusiness.com/real-estate-facebook-page/
18. millions), N. (2019). *Facebook users worldwide 2018 | Statista*. [online] Statista. Available at: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/
19. Weaver, J. (2016). *Your Example Twitter Name - ppt video online download*. [online] Slideplayer.com. Available at: https://slideplayer.com/slide/4495840/
20. millions), N. (2019). *Twitter: number of active users 2010-2018 | Statista*. [online] Statista. Available at: https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/
21. Xu, L. (2015). *6 Examples & Best Practices for Creating Instagram Ads*. [online] Curalate Social Commerce Platform. Available at: https://www.curalate.com/blog/6-examples-and-best-practices-for-creating-instagram-ads/
22. millions), N. (2018). *Instagram: active users 2018 | Statista*. [online] Statista. Available at: https://www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/
23. Koozai.com. (2013). *Creating & Optimising A New YouTube One Channel Design*. [online] Available at: https://www.koozai.com/blog/social-media/video-marketing/creating-optimising-a-new-youtube-one-channel-design/
24. Gilbert, B. (2018). *YouTube now has over 1.8 billion users every month, within spitting distance of Facebook's 2 billion*. [online] Business Insider. Available at: https://www.businessinsider.com/youtube-user-statistics-2018-5?r=UK
25. Edmond, D. (2015). *10 Examples of Highly Impactful LinkedIn Profiles*. [online] KoMarketing: B2B Search, Social, & Content Marketing. Available at: https://komarketing.com/blog/10-examples-highly-impactful-linkedin-profiles/
26. DARROW, B. (2017). *http://fortune.com*. [online] Fortune. Available at: http://fortune.com/2017/04/24/linkedin-users/
27. KALLAS, P. (2012). *41 Great Examples of Pinterest Brand Pages*. [online] Dreamgrow.com. Available at: https://www.dreamgrow.com/41-great-examples-of-pinterest-brand-pages/
28. millions), N. (2018). *Pinterest: monthly active users 2018 | Statistic*. [online] Statista. Available at: https://www.statista.com/statistics/463353/pinterest-global-mau/
29. MakeUseOf. (2013). *The Unofficial, Beginner's Guide to Tumblr*. [online] Available at: https://www.makeuseof.com/tag/the-unofficial-beginners-guide-to-tumblr/
30. Yarow, J. (2013). *The Truth About Tumblr: Its Numbers Are Significantly Worse Than You Think*. [online] Business Insider. Available at: https://www.businessinsider.com/tumblrs-active-users-lighter-than-expected-2013-5?IR=T
31. Patel, N. (n.d.). *Are Snapchat Ads Worth Your Investment? Here's How to Find Out*. [online] Neil Patel. Available at: https://neilpatel.com/blog/explore-snapchat-ads/
32. millions), N. (2019). *Snapchat daily active users 2018 | Statistic*. [online] Statista. Available at: https://www.statista.com/statistics/545967/snapchat-app-dau/#0
33. LLP, P. (2011). *Plans and pricing : PythonAnywhere*. [online] Pythonanywhere.com. Available at: https://www.pythonanywhere.com/pricing/
34. Amazon Web Services, Inc. (2019). *AWS Free Tier*. [online] Available at: https://aws.amazon.com/free/?nc2=h_ql_pr&awsf.Free%20Tier%20Types=categories%23featured
35. Openshift.com. (2019). *OpenShift Product Pricing - Red Hat OpenShift*. [online] Available at: https://www.openshift.com/products/pricing/
36. Heroku.com. (2019). *Pricing | Heroku*. [online] Available at: https://www.heroku.com/pricing#postgres-pricing
37. Goel, A. (n.d.). *Python vs PHP in 2019 - Comparison, Features & Applications*. [online] Hackr.io Blog. Available at: https://hackr.io/blog/python-vs-php-in-2019

38. Flask.pocoo.org. (2010). *SQLAlchemy in Flask — Flask 1.0.2 documentation*. [online] Available at: http://flask.pocoo.org/docs/1.0/patterns/sqlalchemy/
39. Untevskiy, D. (2017). *Flask vs Django. Which Is Better for Your Web App? - Gearheart*. [online] Gearheart.io. Available at: https://gearheart.io/blog/flask-vs-django-which-is-better-for-your-web-app/
40. Tezer, O. (2014). *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems | DigitalOcean*. [online] Digitalocean.com. Available at: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems
41. Creately.com. (2008). *Marketing Web Sitemap Template | Editable Site Map Template on Creately*. [online] Available at: https://creately.com/diagram/example/go2ewdki1/Marketing+Web+Sitemap+Template
42. Balsamiq.com. (2008). *Balsamiq Wireframes | Balsamiq*. [online] Available at: https://balsamiq.com/wireframes/
43. Erdplus.com. (2015). *ERDPlus*. [online] Available at: https://erdplus.com/#/
44. Www1.udel.edu. (n.d.). *DDL: CREATE TABLE*. [online] Available at: http://www1.udel.edu/evelyn/SQL-Class1/SQLTable1.html
45. Techonthenet.com. (2003). *SQL: DDL/DML for Tutorial (INSERT Statement)*. [online] Available at: https://www.techonthenet.com/sql/insert_ddl.php
46. Study.com. (2018). *Writing Pseudocode: Algorithms & Examples - Video & Lesson Transcript | Study.com*. [online] Available at: https://study.com/academy/lesson/writing-pseudocode-algorithms-examples.html
47. Djangoproject.com. (2005). *Django overview | Django*. [online] Available at: https://www.djangoproject.com/start/overview/
48. Atom. (n.d.). *A hackable text editor for the 21st Century*. [online] Available at: https://atom.io/
49. Docs.djangoproject.com. (2005). *Managing static files (e.g. images, JavaScript, CSS) | Django documentation | Django*. [online] Available at: https://docs.djangoproject.com/en/2.1/howto/static-files/
50. Docs.djangoproject.com. (2005). *Models | Django documentation | Django*. [online] Available at: https://docs.djangoproject.com/en/2.1/topics/db/models/
51. Verma, A. (2018). *Learn Django - The Easy Way | Adding Comments | Tutorial - 44*. [online] YouTube. Available at: https://youtu.be/An4hW4TjKhE
52. Goodridge, M. (2017). *maxg203/Django-Tutorials*. [online] GitHub. Available at: https://github.com/maxg203/Django-Tutorials/blob/master/home/models.py
53. Entrepreneurs, C. (2016). *Try Django 1.9 - 33 of 38 - Associate User to Post with a Foreign Key*. [online] YouTube. Available at: https://youtu.be/2h57cqFRcqg
54. Tutorial.djangogirls.org. (n.d.). *Template extending · Django Girls Tutorial*. [online] Available at: https://tutorial.djangogirls.org/en/template_extending/
55. Freitas, V. (2016). *How to Use Django's Built-in Login System*. [online] Simple is Better Than Complex. Available at: https://simpleisbetterthancomplex.com/tutorial/2016/06/27/how-to-use-djangos-built-in-login-system.html
56. Goodridge, M. (2016). *Custom Django User Registration Form (Django Tutorial) | Part 16*. [online] YouTube. Available at: https://www.youtube.com/watch?v=66l9b2QrBR8&list=PLw02n0FEB3E3VSHjyYMcFadtQORvl1Ssj&index=16
57. Stack Overflow. (2013). *Check if a string contains a number*. [online] Available at: https://stackoverflow.com/questions/19859282/check-if-a-string-contains-a-number
58. Parker, J. (2019). *Full List of Bad Words and Swear Words Banned by Google*. [online] Free Web Headers. Available at: https://www.freewebheaders.com/full-list-of-bad-words-banned-by-google/
59. Goldberg, C., Appalaraju, S. and Ransom, M. (2010). *How do you read a file into a list in Python?*. [online] Stack Overflow. Available at:

https://stackoverflow.com/questions/3925614/how-do-you-read-a-file-into-a-list-in-python/3925701
60. Docs.djangoproject.com. (2005). *Form and field validation | Django documentation | Django*. [online] Available at: https://docs.djangoproject.com/en/2.1/ref/forms/validation/
61. Goodridge, M. (2017). *maxg203/Django-Tutorials*. [online] GitHub. Available at: https://github.com/maxg203/Django-Tutorials/blob/master/accounts/models.py
62. Goodridge, M. (2017). *How to Display UserProfile Model Info in Templates on Profile Page (Django Tutorial) | Part 35*. [online] YouTube. Available at: https://www.youtube.com/watch?v=2yoWf-kDXIk&list=PLw02n0FEB3E3VSHjyYMcFadtQORvl1Ssj&index=35
63. Entrepeneurs, C. (2016). *Try Django 1.9 - 37 of 38 - Search Posts*. [online] YouTube. Available at: https://www.youtube.com/watch?v=eyAIAZr5Q3w&index=13&list=WL&t=0s
64. Entrepreneurs, C. (2016). *Try Django 1.9 - 17 of 38 - Get Item or 404 Query*. [online] YouTube. Available at: https://www.youtube.com/watch?v=zg75OzTVOnU&list=PLEsfXFp6DpzQFqfCur9CJ4QnKQTVXUsRy&index=17
65. Verma, A. (2018). *Learn Django - The Easy Way | Creating a Like/Dislike Button | Tutorial - 37*. [online] YouTube. Available at: https://www.youtube.com/watch?v=VoWw1Y5qqt8&list=WL&index=3&t=0s
66. Entrepreneurs, C. (2016). *Try Django 1.9 - 23 of 38 - Delete View*. [online] YouTube. Available at: https://www.youtube.com/watch?v=zwPsFIVMRX0&list=WL&index=13
67. Goodridge, M. (2017). *How to Present a Friends List on the Home Page in Django (Django Tutorial) | Part 58*. [online] YouTube. Available at: https://www.youtube.com/watch?v=ZGbYmMF4QLI&list=WL&index=6
68. Goodridge, M. (2017). *How to Add Bootstrap Buttons to Finish the Friends List (Django Tutorial) | Part 59*. [online] YouTube. Available at: https://www.youtube.com/watch?v=wLZlyjq_mFc&list=WL&index=7
69. Goodridge, M. (2017). *How to Create Useful Many to Many Relationships in Django (Django Tutorial) | Part 56*. [online] YouTube. Available at: https://www.youtube.com/watch?v=bFhuOULgKDs&list=WL&index=8
70. Goodridge, M. (2017). *How to Use a Many to Many Field in a Django Model (Django Tutorial) | Part 55*. [online] YouTube. Available at: https://www.youtube.com/watch?v=IXJ46DitsIg&list=WL&index=9
71. Goodridge, M. (2017). *How to Create a Many to Many Relationship Between Django Model Objects (Django Tutorial) | Part 54*. [online] YouTube. Available at: https://www.youtube.com/watch?v=nwpLCa79DUw&list=WL&index=10
72. Goodridge, M. (2017). *How to Pass a Primary Key through a Django URL (Django Tutorial) | Part 53*. [online] YouTube. Available at: https://www.youtube.com/watch?v=zcJegVlKqqs&list=WL&index=11
73. Cherian, J. and Afzal, V. (2012). *How to show all fields of model in admin page?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/10543032/how-to-show-all-fields-of-model-in-admin-page
74. Medium. (2017). *Some Cool Things You Can Do With Python*. [online] Available at: https://medium.com/@Intersog/some-cool-things-you-can-do-with-python-12a595164b49
75. Barton, C. (n.d.). *Solving Equations with Surds: GCSE Maths Question of the Week (Higher) - Mr Barton Maths Blog*. [online] Mr Barton Maths Blog. Available at: http://www.mrbartonmaths.com/blog/solving-equations-with-surds-gcse-maths-question-of-the-week-higher/