Computer Science 334

Web Development
Practical Report

Social Media Question and Answer Website

Contributors:	*Study:*	*Student Numbers:*
Pieter Goos	EE	19231466
Richard Batt	EE	19116683
Jason George	EE	18956122
Michael Cook	CS	17794633
Simba Chawatama	EE	19030827
Christiaan Meyer	CS	19007361

May 12, 2018

Universiteit iYunivesithi Stellenbosch University 100 1918 · 2018

saam vorentoe · masiye phambili · forward together

Plagiarism Declaration

1. Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as our own.
2. We agree that plagiarism is a punishable offence because it constitutes theft.
3. We also understand that direct translations are plagiarism.
4. Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. We understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
5. We declare that the work contained in this assignment, except where otherwise stated, is our original work and that we have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

[illegible]	[illegible]
Pieter Goos	Richard Batt
[illegible]	[illegible]
Simba Chawatama	Michael Cook
[illegible]	[illegible]
Jason George	Christiaan Meyer

Contents

List of Figures

List of Tables

1 Introduction

This report will cover both the design and implementation of Python and Graph Databases into a website. In this case Flask - a python 3 package - and Neo4j are used. This website has to take the form of Question and Answers website - similar to Quora.com. Detailed descriptions of the Queries in Neo4j's Cypher language, Page Designs and Flask implementation will be covered in their respective sections. Furthermore the website will be criticized for its flaws and possible alternatives that could have been used to run the Database.

2 Database Design

Neo4j is used for this website. This is a graph database management system allowing for relationships and nodes to interact with one-another. The design of the nodes, relationships and cypher queries will be discussed in this section.

2.1 Nodes and Relationships

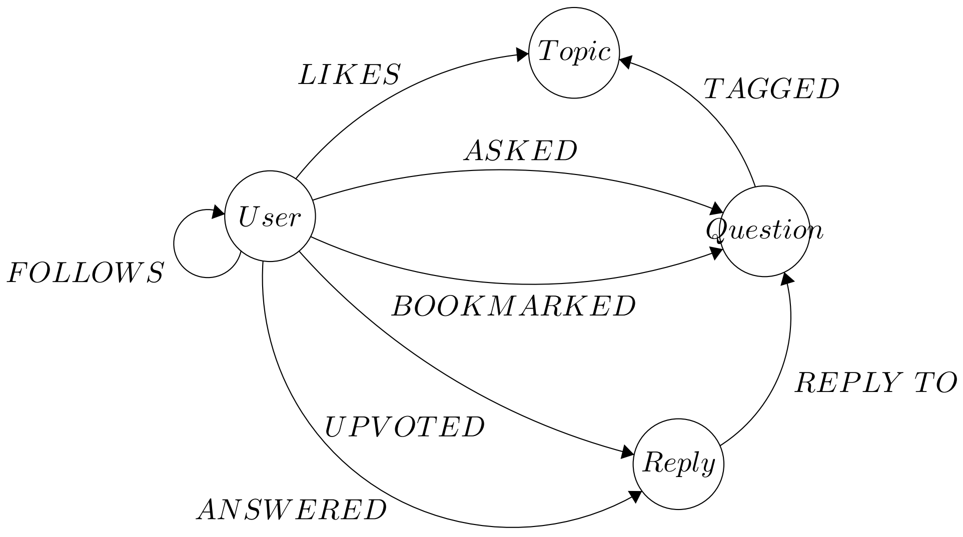


Figure 1: Neo4j Nodes and Relationships

Graph databases allow for nodes with properties to be created and then related together. As can be seen in Figure 1 there are only 4 nodes required with 8 relationships between them all. This setup is simple yet contains enough information for the system to work. Each node has properties assigned to it as can be seen in Table 1. In this section nodes will be denoted by n and relationships by r.

Topic	User	Question	Reply
topic	username	id	id
	fullName	title	text
	bio	text	date
	password	date	
	email		

Table 1: Contents of Nodes

2.1.1 n:Topic

This node contains one property, named topic, which contains the plain-text name of all the predetermined topics. In the case of this project the topics were selected to be:

- Psychology
- Travel
- Entertainment
- Food
- Hobbies

- Nightlife
- Science

These topics were selected as to have a reasonable selection of general enough topics to either show interest in as a user or to use as a tag for a question. To create a more specific topic users can combine multiple topics from the list to allow for others to get a better idea of the intended topic.
Topics do not have any relationships originating from them.

2.1.2 n:User

The user node is the node with the most information. This node contains all the information relating to each individual user. Upon registration the user must supply a username (An alphanumeric string with no special characters), Full Name, password and email address. Later, on the user's own page, the user can change his/her bio which is a short optional self-description. The password which is entered on the homepage's registration form is encrypted by using the bcrypt python package before being stored in the User node. The User node is the only node which has its data edited (in Cypher this is called setting). The website allows for the bio and password to be changed through the user's profile editing menu.
The User node can be the source of many relationship types: r:LIKES, r:ASKED, r:BOOKMARKED, r:UPVOTED, r:ANSWERED and r:FOLLOWS.
r:LIKES is used to denote when the user shows interest in a certain topic node. This information can later be used for finding posts the user may want to see.
r:ASKED shows the correlation of a user to a question, i.e. a user asked a question. This is later used to display the question's original author and the relevant user information.
r:BOOKMARKED links the user to a question they would like to keep a close eye on. This relationship is only used to display bookmarked posts on the user's personal profile page.
r:UPVOTED allows for certain answers to receive a better rating than others, allowing for the most 'upvoted' answers to be shown first on the question page. The reason for this relationship to come from a User is to make it simpler to not allow for multiple upvotes from one account.
r:ANSWERED is a relationship similar to r:ASKED where it just links the reply node to a user as to show the relevant author information.
r:FOLLOWS links one user to another to, similarly to liking a topic, show posts by users that the user follows. This is also used to hide certain posts when visiting another user's profile when that user is not being followed already.

2.1.3 n:Question

The question node contains all the relevant information regarding the question being posed. Upon creating the question the user must fill in a title and the body of the question. The python function then obtains the date automatically and generates a unique UUID4 number to use as the id. The id will later be used to reference to the question's page. The date will be used when displaying the question's thread along side that of others, these will then all be sorted by which has been posted most recently. Note however that in certain situations if the question has a reply the date of the reply will replace that of the question.
A question node can have the r:TAGGED relationship indicating the topics that the user, at creation, decided would be applicable to the question.

2.1.4 n:Reply

Finally, the reply node is very similar to the n:Question node as it is essentially a subsidiary of the question node. The reply node has a text value which stores the answer that a user has given. Other than this, like in the question node, the reply node has a UUID4 id and date automatically assigned to it. Replies are only visible on its question's page, essentially making a question like a thread housing all its answers.

The reply node only has the r:REPLYTO relationship. This relationship is always present as it links the reply to its question node, as mentioned prior.

2.2 Cypher Queries

In Neo4j the queries are in Cypher. This section will cover a large portion of the Cypher queries used to achieve certain tasks. Each query will have its purpose explained. Note that in this section when curly braces, { and }, are used that these indicate variables with the name in the center. This variable will be clarified for each specific case as these may not be unique between queries.

2.2.1 Basic Relationship Check

```
MATCH (u:User)-[:LIKES]->(n:Topic)
WHERE u.username={user}
RETURN n
```

This Cypher query will return all topics which a user with the username specified in the user variable has the LIKES relationship.

2.2.2 Adding and Removing relationships

```
MATCH (u:User), (t:Topic)
WHERE u.username={user} AND t.topic={top}
CREATE (u)-[:LIKES]->(t)

MATCH(u:User)-[r:LIKES]-(t:Topic)
WHERE u.username={user} AND t.topic={top}
DELETE r
```

These two queries add and remove relationships respectively. In this case these queries will cause a user to have the LIKES relationship with a topic or to remove the relationship. for each query the variable user and top are supplied, these being the username and topics respectively.

2.2.3 Editing a Node

```
MATCH(u:User)
WHERE u.username={user}
SET u.bio = {bio}
RETURN u
```

This query shows the bio of a certain user with the username variable user. The bio is then set to have the same content as that of the variable of the same name.

2.2.4 Counting upvotes

```
MATCH (u:User)-[:ANSWERED]-(r:Reply)
WHERE u.username = {user}
OPTIONAL MATCH (b:User)-[e:UPVOTED]-(r)
RETURN count(b) AS cnt, r
```

This query counts the number of upvotes that a certain user has obtained from people upvoting his/her answers.

2.2.5 Obtaining Suggested Follow

```
MATCH (u:User), (i:User)
WHERE i.username = {user} AND NOT u.username = {user} AND NOT (u)<-[:
    FOLLOWS]-(i)
OPTIONAL MATCH (c:User)-[:UPVOTED]->(r:Reply)
WHERE (u)-[:ANSWERED]->(r)
RETURN u, count(c) as cnt ORDER BY cnt DESC LIMIT 5
```

The query obtains a list of users that cannot contain itself which the user does not already follow. This counts as the suggested users to follow. This query will at most only show 5 suggestions at once and is sorted by said person's upvotes in descending order.

2.2.6 Checking Existence

```
MATCH(u:User), (q:User)
WHERE u.username={user} AND q.username={text}
RETURN EXISTS((u)-[:FOLLOWS]->(q))
```

This query returns a boolean value of whether a certain relationship exists. In this case if a certain user already follows another. Variables represent the respective usernames.

2.2.7 Latest Questions and Answers

```
MATCH (u:User)-[:ASKED]->(q:Question)
WHERE u.username={user}
RETURN q as out
UNION
MATCH (u)-[:ANSWERED]->(r:Reply)
WHERE u.username={user}
RETURN r as out
ORDER BY out.date DESC LIMIT 10
```

This query obtains a user's latest 10 questions and answers and ranks them by date in a single list. This makes use of the UNION command to link essentially 2 different queries together into one result.

2.2.8 Searching

```
MATCH(n:Question)
WHERE toLower(n.title) CONTAINS toLower({quer})
RETURN n ORDER BY n.date

MATCH(n:User)
WHERE toLower(n.username) CONTAINS toLower({quer}) AND NOT n.username={
    slf}
RETURN n
```

These queries return questions and users, respectively, containing a certain search term. The user search does not show the current user. The results for the questions are sorted by date.

2.2.9 Get Replies

```
MATCH (n:Reply)-[:REPLYTO]->(m:Question)
WHERE m.id = {quer}
```

```
OPTIONAL MATCH (u:User)-[r:UPVOTED]-(n:Reply)
RETURN n, COUNT(u) ORDER BY COUNT(u) DESC
```

This query returns all the replies to a certain question ordered by the number of upvotes each reply received.

2.2.10 Threads the user may like

```
MATCH (q:Question), (me:User), (th:User), (t:Topic)
WHERE me.username={username} AND ((me)-[:FOLLOWS]->(th) AND ((th)-[:ASKED
    ]->(q) OR (th)-[:ANSWERED]->(:Reply)-[:REPLYTO]->(q))) OR ((q)-[:
    TAGGED]-(t) AND (me)-[:LIKES]-(t))
OPTIONAL MATCH (b:User)-[uu:UPVOTED]-(re:Reply)
WHERE (re)-[:REPLYTO]-(q)
RETURN q, (CASE WHEN MAX(re.date) > (q.date) THEN MAX(re.date) ELSE q.
    date END) AS cnt ORDER BY cnt DESC

MATCH (q:Question), (me:User), (th:User), (t:Topic)
WHERE me.username={username} AND ((me)-[:FOLLOWS]->(th) AND ((th)-[:ASKED
    ]->(q) OR (th)-[:ANSWERED]->(:Reply)-[:REPLYTO]->(q))) OR ((q)-[:
    TAGGED]-(t) AND (me)-[:LIKES]-(t))
OPTIONAL MATCH (b:User)-[uu:UPVOTED]-(re:Reply)
WHERE (re)-[:REPLYTO]-(q)
RETURN q, count(DISTINCT(b)) AS cnt ORDER BY cnt DESC
```

These queries return question threads which the user has either shown interest in the same topic(s) or followed the user that posted the question or reply. The first orders the results by the latest date of either the question or reply, the second by upvotes. Note however these queries seem to not work 100% correctly.

2.2.11 Follow of a Follow

```
MATCH (q:Question), (me:User), (th:User), (t:Topic)
WHERE me.username={username} AND ((me)-[:FOLLOWS]->(:User)-[:FOLLOWS]->(
    th) AND ((th)-[:ASKED]->(q) OR (th)-[:ANSWERED]->(:Reply)-[:REPLYTO
    ]->(q)))
OPTIONAL MATCH (b:User)-[:UPVOTED]-(re:Reply)
WHERE (re)-[:REPLYTO]-(q)
RETURN q, count(DISTINCT(b)) AS cnt ORDER BY cnt DESC LIMIT 10
```

If the user is to follow another user, this query will return the questions that said user follows. This is then ranked by the number of upvotes a question thread has received and limited to the top 10 results.

3 Page Design

3.1 Navbar

This website makes use of the Bootstrap navbar. For this project the navbar has two forms, one on the homepage and one for once the user has logged in.
When the user is on the homepage, which is discussed further in a later section, the navbar is home

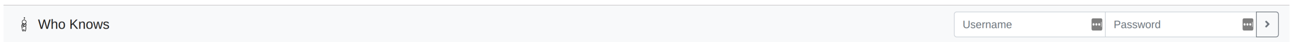


Figure 2: Hompage Navbar

to the logo and the login form. This login form is simply a text field and a password field with a submit button. This can be seen in Figure 2.
On all other pages the navbar looks as in Figure 3. This navbar houses the logo on the left, acting as

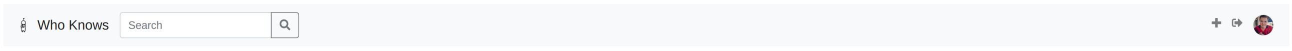
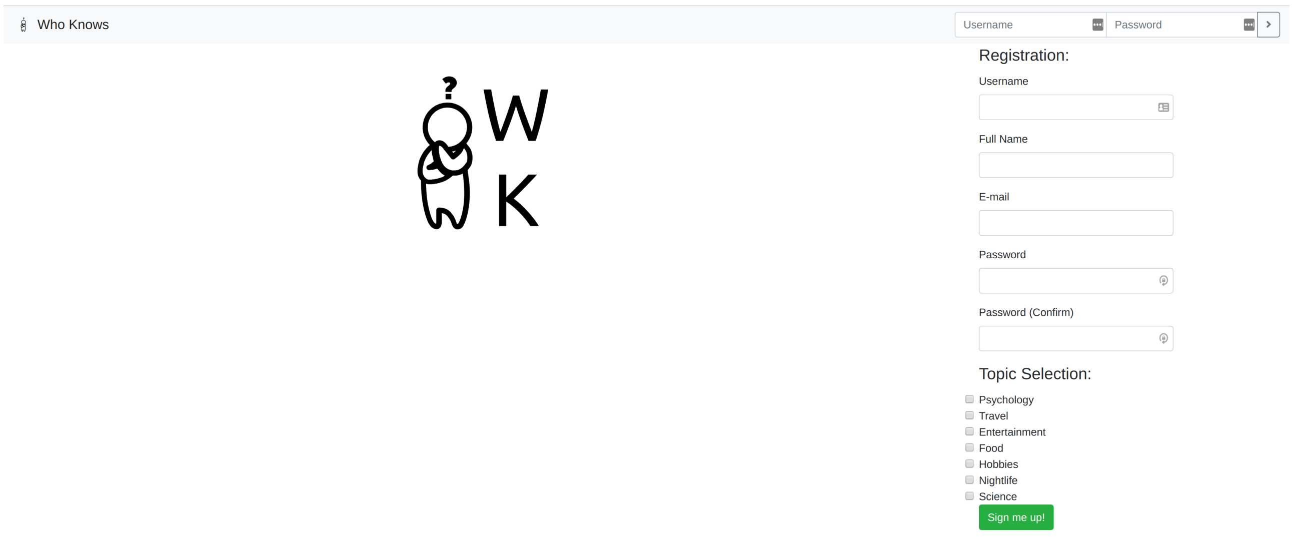


Figure 3: General Navbar

a hyperlink to the user's homepage. To the right of the logo there is the search box and button, this will be discussed in a later section. Finally, on the right, multiple buttons show, these include adding a post (+), Logging out the user, and the user's profile (their profile picture). When the user is on their own profile an edit button will appear allowing the user to edit some of their information.
The navbars make use of icons rather than text as to give a cleaner appearance. Each icon does have a description if the user hovers their cursor over it. The site is also completely responsive and will collapse the navbar into a drop down list when the display is small enough. The descriptions for each button are then simply shown next to the icon.
In following figures and sections the navbar may not be shown or discussed any further.

3.2 Homepage

localhost:5000/



Registration:

Username

Full Name

E-mail

Password

Password (Confirm)

Topic Selection:

- Psychology
- Travel
- Entertainment
- Food
- Hobbies
- Nightlife
- Science

Sign me up!

Figure 4: Homepage

The homepage houses only the registration form for new users as shown in Figure 4. This form takes in the username, full name, email address, password (with confirmation) and topic selection. This form is submitted via a POST requirement to Flask. The form will not submit if the validation on the email, username and password fields are not met. The username must be alphanumeric, the email must be of the correct format with @ and ., finally, the password must be at least 8 characters long, contain one capital letter, one lower case letter and a number.
The login form, in the navbar, does not have any validation on it as it is expected that the user will spell their username and password correctly.

3.3 Profile

localhost:5000/p/<username>

The profile page is essentially a template for three separate modes. Two of which are when the user which is currently logged in and viewing his/her own profile. The other view is when the user is viewing another user's profile. Each of these views will be discussed below. The general look is to have the user's information on the left of screen and to have posts, bookmarks and other links on the right.

3.3.1 Own Profile (Private Mode)

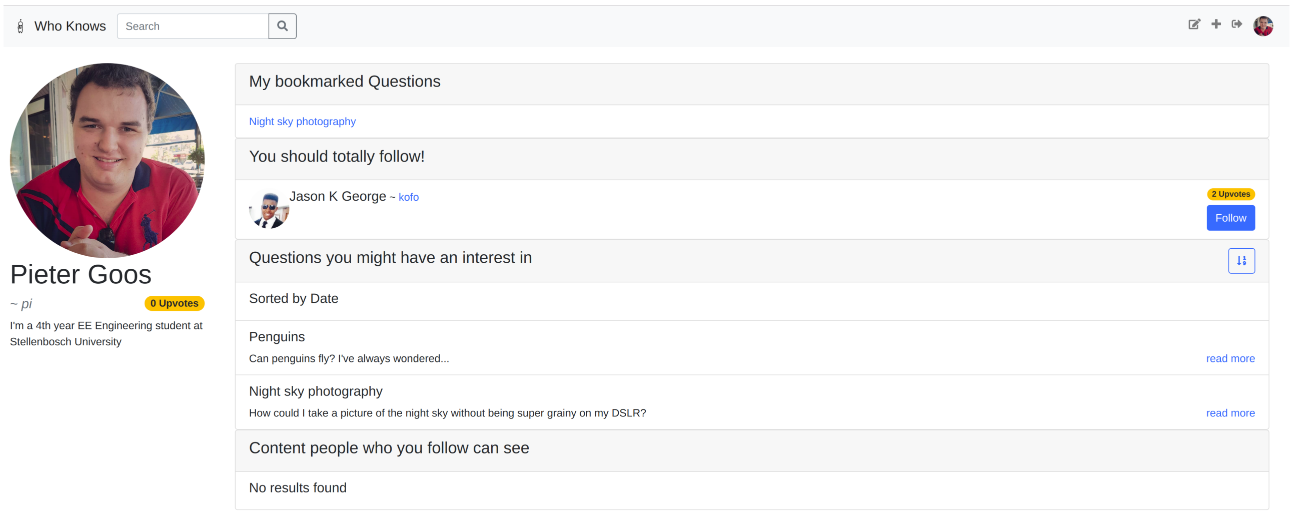
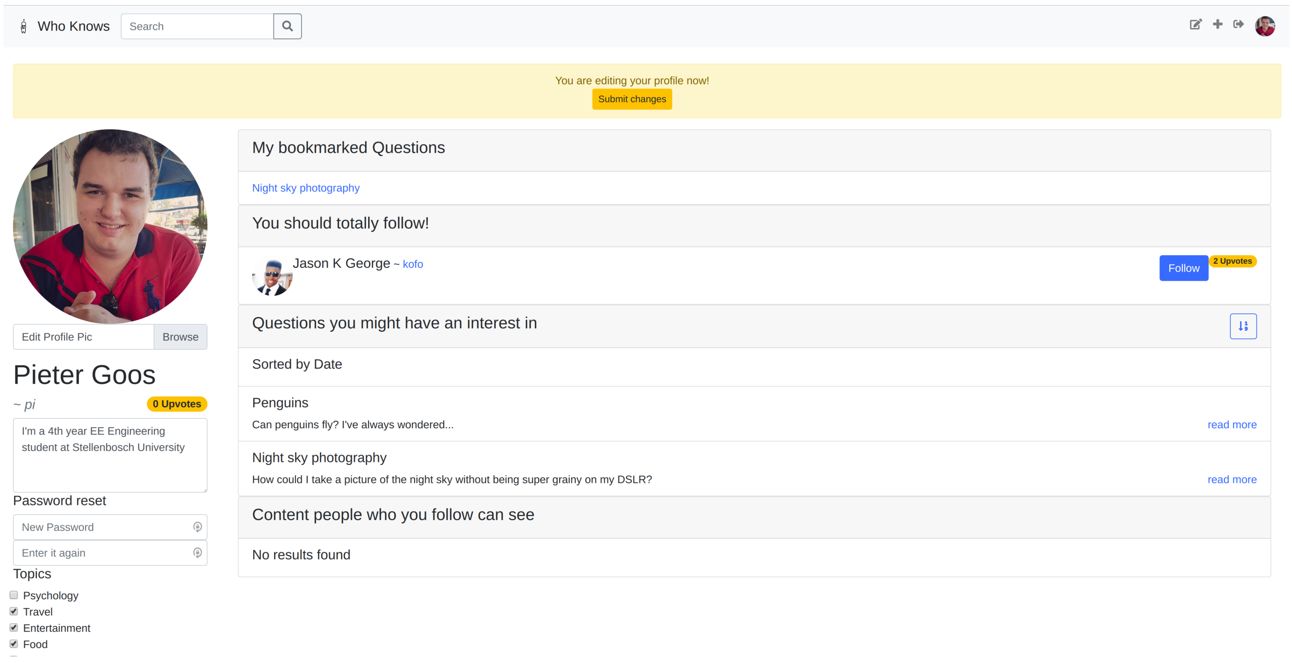

Who Knows
Search
Pieter Goos
~ pi
0 Upvotes
I'm a 4th year EE Engineering student at Stellenbosch University
My bookmarked Questions
Night sky photography
You should totally follow!
Jason K George ~ kofo
2 Upvotes
Follow
Questions you might have an interest in
Sorted by Date
Penguins
Can penguins fly? I've always wondered...
read more
Night sky photography
How could I take a picture of the night sky without being super grainy on my DSLR?
read more
Content people who you follow can see
No results found

Figure 5: One's own Profile Page

When the user is logged in they will be redirected to their own profile page. The user's own page can also be accessed from any point on the website by pressing any one of the shortcuts when the user is referenced. In this view the user will see their profile image, name and username on the left above their bio and number of upvotes. This number is determined by counting the amount of upvotes that answers which the user posted received. In the right pane the other information will be displayed. This includes the bookmarked questions. This is a simple list in a Bootstrap card of all the questions the user has marked with a bookmark. This relationship is set on the question's page. Below this a list of users is shown, these are users that he/she may want to follow. This is determined by listing users which they do not yet follow and is sorted by number of upvotes they received. Another card is found below the prior, this one contains a list of questions which the user may have an interest in. This metric is decided by showing questions and answers which are tagged as topics that the user follows or which are written by a user that they follow. The sorting method can be changed by pressing the button on the right of the title to this section. The two options are by date or by amount of upvotes, the current sorting type is shown above all the answers. Finally there is one more card which shows the content which the users that the user follows can see. All these cards will show a warning when there are no results (as can be seen at the bottom of Figure 8). The only exception to this is the bookmarks list, this simply does not show if there are no bookmarked questions.

3.3.2 Editing Profile

This view is a slight modification from the Private Profile page as seen in Figure 6. This is due to the only access to this page being from that page. The right hand column will be identical, however the left will look slightly different. At the top of the page there is a yellow banner bar with the form submission button. The left column now houses a field to upload a new .png file as the user's profile


Who Knows

Search

You are editing your profile now!

Submit changes

Edit Profile Pic | Browse

Pieter Goos

~ pi 0 Upvotes

I'm a 4th year EE Engineering student at Stellenbosch University

Password reset

New Password

Enter it again

Topics

- [ ] Psychology
- [x] Travel
- [x] Entertainment
- [x] Food

My bookmarked Questions

Night sky photography

You should totally follow!

Jason K George ~ kofo Follow 2 Upvotes

Questions you might have an interest in

Sorted by Date

Penguins

Can penguins fly? I've always wondered... read more

Night sky photography

How could I take a picture of the night sky without being super grainy on my DSLR? read more

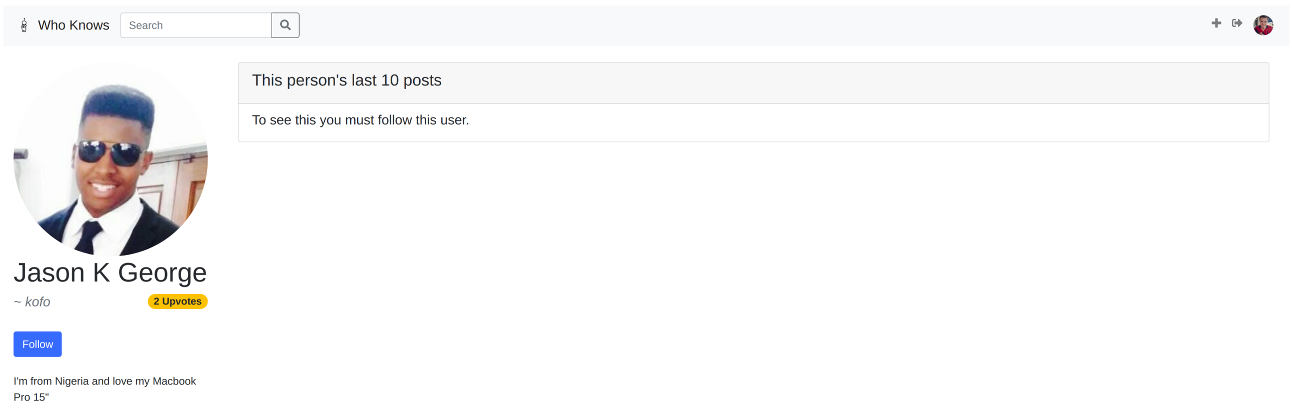
Content people who you follow can see

No results found

Figure 6: Editing Profile Page

picture. The user can also change his/her bio here, as well as changing their password. Finally, they can change their followed topics. None of the fields are compulsory and the submission will use a POST request to submit the changes, if any, to the server.

3.3.3 Other's Profile (Public Mode)


Who Knows

Search

Jason K George

~ kofo 2 Upvotes

Follow

I'm from Nigeria and love my Macbook Pro 15"

This person's last 10 posts

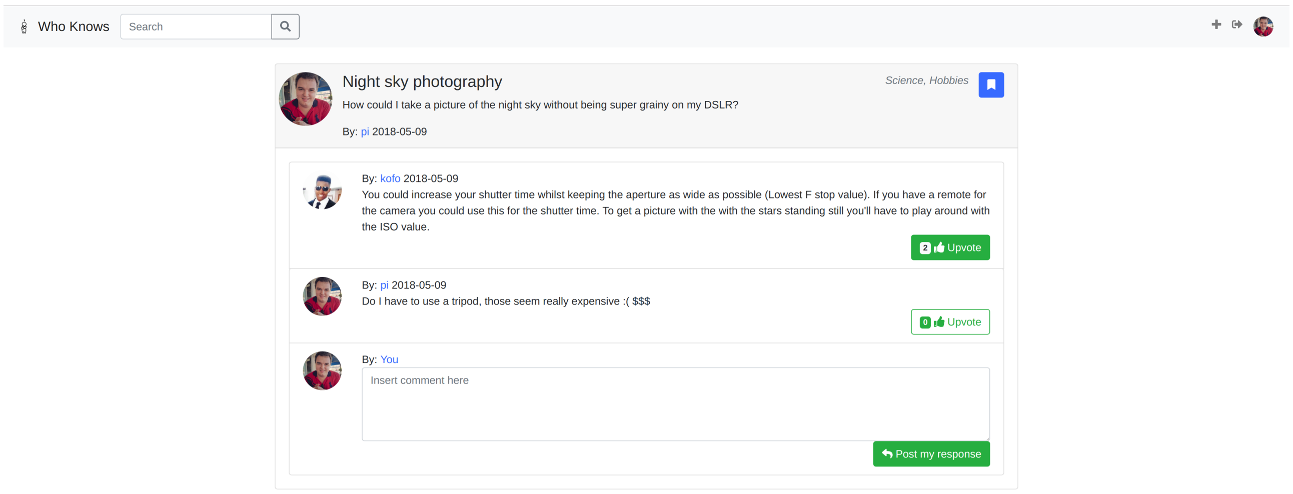
To see this you must follow this user.

Figure 7: Another user's profile page

This is the page that one can see if they visit another user's profile page. An example of this can be seen in Figure 7. This page, like the private profile page, shows the user's information on the left. In this view however there is a follow/unfollow button that the current user can use to create said relationship in the database. The right hand column shows only one card. This card contains the user's latest ten questions and answers - sorted by date. This information is only shown if the current user is following the user whose profile is being viewed. The Follow button makes use of a POST request to update the relationship.

3.4 Question

localhost:5000/q/<id>



Night sky photography

Science, Hobbies

How could I take a picture of the night sky without being super grainy on my DSLR?

By: pi 2018-05-09

By: kofo 2018-05-09

You could increase your shutter time whilst keeping the aperture as wide as possible (Lowest F stop value). If you have a remote for the camera you could use this for the shutter time. To get a picture with the with the stars standing still you'll have to play around with the ISO value.

By: pi 2018-05-09

Do I have to use a tripod, those seem really expensive :($$$

By: You

Figure 8: Question Viewing Page

The question viewing page, as seen in Figure 8, houses a question with all of its replies. The question and each answer shows the author, the date it was authored and their profile picture along side the text. Toggle-able buttons are used to show the state of the user bookmarking the question and upvoting the answers. The replies are all kept in cards below the question title, which is highlighted in a light grey. Each username shown on the page is clickable and will take you to said user's profile page. The field where a user can add a new reply is always present and thus if no answers have already been given the this will be the only 'reply'. This field houses a text area and a button to POST the content of the reply to the website. As mentioned in the database design, the reply will automatically link to the question and self populate fields like id and date.

3.5 Add Question

localhost:5000/addQ

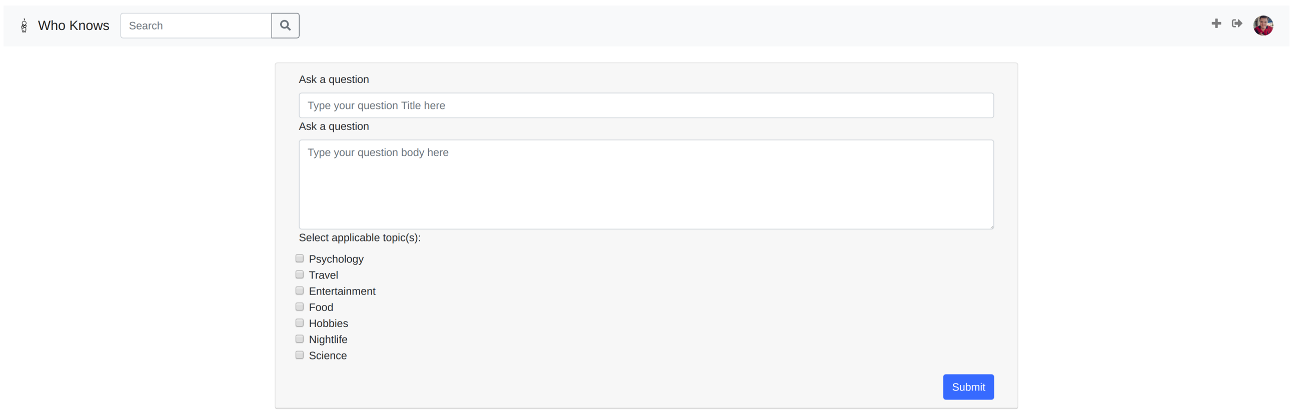


Figure 9: Question Adding Page

The question adding page, as can been seen in Figure 9, is a simple form which allows for any user to add a new question to the site's database. Only three components are needed for this section, The question title, the question text body and which topics the user finds relevant. The user is simply required to fill in all three inputs and then submit, using a POST request which is then used in Flask. Once you submit your question the page will automatically redirect to the question you have just posted.

3.6 Search Results

localhost:5000/s/<query>

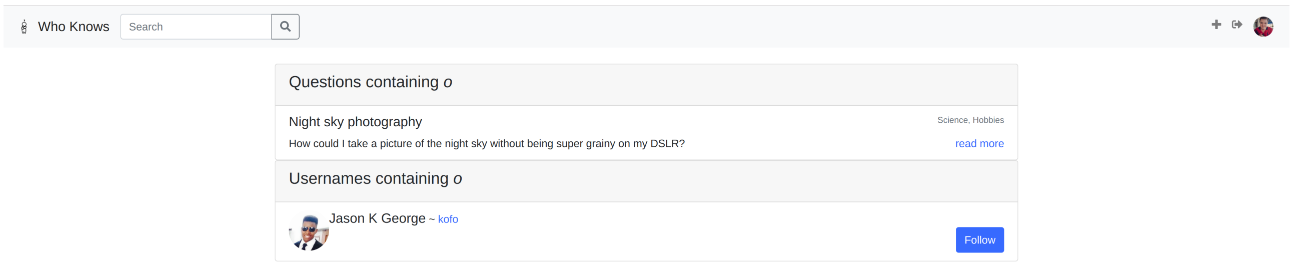


Figure 10: Search Results Page

The search page, as can be seen in Figure 10, houses the questions and users whose titles and usernames respectively contain a certain search term, in this case 'o'. The results each get put into another Bootstrap card element underneath their appropriate title. The search algorithm only checks the titles of the questions and not the text body as this seemed like a more granular approach. Each question will show the title, text body and its topic(s) with a link to said question. For the users the usernames are searched. The user's profile picture will be shown with their full name and username. The username acts as a hyperlink to the user's profile page. The user can also be followed from this page, this is achieved with a POST request and thus requires the page to be reloaded (this is automatic).

4 Flask and Neo4j Implementation

For this project the Flask microframework for python 3 was used. Flask is based off of Werkzeug and Jinja 2. To store all the data a Neo4j Graph based database was used. These two systems were linked together using py2neo. Furthermore the bcrypt, shutil and passlib libraries are used. The file structure was based off of the supplied example by Nicole White [1].

4.1 run.py

This is the python file which starts the Flask app and serves no other purpose other than to change the launch settings.

4.2 __init__.py

This file automatically runs when the server starts and runs database commands. The database is initialized by checking if the Topics are in place already, if not, they are added. Constraints on all node types are added to prevent duplicates of values that are expected to be unique. The *create_-unique_constraint()* function is taken from Nicole White's guide [1].

4.3 views.py

This file is split into functions, one per page of the website. The section titles below will be the same as the function names in the views.py file.

4.3.1 index

Upon receiving a GET request this function will check if a user is already logged in. If this is the case then they will be redirected to their profile page, otherwise the homepage template will be rendered. When a POST request is received the function will check whether it is to log in or register the user. For logging in a simple check of the username and password is completed redirecting to the profile page if successful. For registering a new user the passwords will be compared as they need to be the same. After this form is retrieved and then sent to the appropriate function in the models python file. If the user is added successfully they will automatically be logged in. Each check completed can produce an error for the user in case of a mistake on their part.

4.3.2 profile

This function will produce the profile page of a user which is specified in the URL. The function checks if the current user is viewing their own profile, and if so adjustments will be made as not to call functions pointlessly. All the data that needs to be shown on the page will be generated by calling the appropriate functions in the models.py file. The function can interpret a POST request for two reasons. If the user is trying to follow or unfollow said user of if they are trying to edit their page. If the user is trying to (un)follow another user that function in the models file is called. However, if the user is submitting their changes to their profile the function will resubmit all what was already placed before. Fields such as the bio and topics simply show their current content in edit mode and then are resubmitted. If a file is uploaded that will be moved to a temporary folder and then renamed and deleted once the user's username is assigned to the png for their picture. Finally, if their passwords match the password will be updated to what was entered.

4.3.3 search

Upon receiving a GET request this function will simply obtain all the questions and users which match the search criteria from the database and display them. This function can also handle POST requests. These post requests will be for following or unfollowing a certain user. This is achieved by simply checking which button has been pressed and then executing the appropriate function in the models python file.

4.3.4 question

This function will render the webpage with all the data it receives from the database based on the query upon receiving a GET request. If a POST request is received it will check what is intended to be done by the user, whether it is adding or removing a bookmark or up/down-voting an answer or if they are submitting a reply. For up/down-voting and adding or removing a bookmark the function will simply run a function to make the appropriate change in the database before the page is rendered. If there is a reply being added the text will first be checked to not be empty and if so then it will add a reply to the question in the database.

4.3.5 newquestion

This function renders the new question page when it has received a GET request. However, when a POST request is received it will check if the title and text areas are not empty and then check which topics are selected. From there the function will make a request to the models.py file to add the question node with all its relevant details and relationships to the database. When this has completed the user will be redirected to the newly created question's page.
A mistake was made in this section to not check if zero topics are selected, this could have simply been a python check just as checking for no content in the text boxes was checked.

4.3.6 logout

This function is called when the user has pressed the logout button in their navbar. All this function does is it pops the user's username out of the current session. After which it will redirect the browser to the homepage where a message will display that the user has in fact logged out.

4.3.7 searchH

This function only gets called when the user attempts to search for something. The query is passed as a POST request to this function. This function then simply changes the request into a new url which Flask then interprets as a search term. For example if you are looking for 'hello' then the redirected page would be *localhost:5000/s/hello*.

4.3.8 goHome

This function is called when the user gets a code 500 Internal Server Error. This error is usually caused by the user attempting to access a page without being logged in. The user's browser will simply redirect them to the homepage.

4.3.9 gotLost

This function is called when the user attempts to load that does not exist. The user will have his/her browser redirect to 404.html in the templates folder.

4.4 models.py

The models file is where all the communications between python and the Neo4j database are programmed. In this file the graph data type is initialized as *graph*. There is a class present to house all commands relating to the user (User). This class has an initializer which stores the user's username. Outside of this class there are two functions, one to get the date in a predefined format and one to initialize the database as discussed in a prior subsection. In this file two methods have been used to read and alter the database: running cypher queries and interpreting the result directly and using premade functions from the py2neo library.

4.4.1 py2neo Commands

The py2neo functions aim to allow the developer to not need Cypher queries. Some examples of these functions include the Node() function allowing you to design the contents of a node as a function; and Relationship() allowing you to link two nodes with a Relationship name. Examples of this being used in this project can be found in the addReply(), upvote() and followUser() functions. There are few of these py2neo functions used as they proved to be more complex to format than simply designing and interpreting the Cypher queries.

4.4.2 py2neo Running Cypher Queries

When making more complex requests to the database it is simpler to use cypher queries and to interpret their output. When running these queries you use *graph.run(query).* If there is expected to be a single result you can use *.evaluate().* This will return a String with the result. Otherwise, when there are multiple results expected, one can iterate over the results and then index the iterator to get out the desired result. As an example, which can be seen in gerSearchUser(), you could want the user's **username** from node **n**, this could be obtained by: record['n']['username'] where record is the iterator over the results from running a query.
This method is used for most of this project as it is simpler to design a query in the Neo4j software and then port it to py2neo for use on the website. This also allows full cognition of what the actual query being run is, something the prebuilt functions do not offer.

4.5 HTML Jinja2 Integration

Jinja allows for the HTML files, called templates, to have python commands and variables in them. As well as having python code in the HTML one can extend and include other HTML files. In this project there is a layout.html file used as the base template for all webpages, hence the **extends** command is used at the top of all other pages. The navbar search is treated similarly by using the **include** command.
When calling *render_template()* in the views.py file all variables used by Jinja must be passed through. In the HTML files for this project functions such as *if* and *for* are used. These functions, for example, allow only certain parts of the profile UI to show. The for loops are often used to have one general piece of HTML code, often card parts, repeated for all the results obtained from an array. Making use of Jinja allows for there to be one piece of HTML code that is dynamically regenerated on request to show the desired content. This therefore explains the name *templates.*

5 Relational vs Graph based Databases

Neo4j is a graphical database which means it uses graph structures for queries using nodes and edges to represent and store data.
A relational database such as SQL stores highly structured data in tables with columns that are predetermined to be of a specific type and with rows of the same type of information.

5.1 Reference to other data

With a relational database such as SQL reference data in other tables and rows are made by referring the primary keys of their data using a foreign key column. For many to many relationships a joining table would have to be used to hold the foreign keys.
The neo4j graph database has nodes which hold the different properties. These properties are stored as key value pairs. To provide connection between the different nodes relationships are used which are direct connections between two different nodes.
Using a relational database instead of neo4j would thus require us to use foreign key columns instead of the simple relationships used. This would result in more complex and more resource expensive system.

5.2 Queries

Neo4j uses Cypher which is a declarative graph query language to handle the database queries. Cypher is based on basic concepts of SQL. The syntax is more concise than that of SQL queries.

5.2.1 Creating new Users

Creating a new user in Cypher:

```
CREATE (u:User {username:UserOne, password:12345})
RETURN u
```

Creating a new user in SQL:

```
INSERT INTO User (username, password)
VALUES (userOne, 12345)
```

From the above syntaxes it can be seen that both methods would result in using generally simple and concise code.

5.2.2 Relationship Checks

A cypher query to return topics liked by a user looks like the following:

```
MATCH (u : User)[:LIKES]->(n : Topic)
WHERE u.username = {user}
RETURN n
```

In SQL querying for topics liked by a user would require a much more complex syntax as is shown in the example query below:

```
SELECT topic FROM user
LEFT JOIN User_Topic
ON User.Id = User_Topic.UserId
LEFT JOIN Topic
ON Topic.Id = User_Topic.TopicId
```

Both queries would return all topics that the user likes. Thus having used a relational database would have resulted in having longer query statements.

5.2.3 Editing a Table

In SQL editing a row of data in a table would correspond to editing a node in neo4j. Using the relational database the syntax would be as below:

```
UPDATE User
SET bio = Insert a bio update
WHERE username = UserOne
```

The syntax is quite similar to the one for editing a node in neo4j.

5.3 Conclusion

The above examples comparing the relational database (SQL) to the graphical database used (neo4j) show that for our application where relationships are important a relational database would have been much difficult to implement due to the use of foreign key columns. Some of the syntax would be similar between the two databases.

A Project Contributions

A.1 Table of Contributions

Contributor	Contributions
P. Goos EE (Project Leader)	Basic Page Design Database Design Neo4j Query (Cypher) Design All Neo4j (py2neo) Implementation Majority of Flask Implementation Designed Navbar Set up file structure with instructions Wrote Majority of Report Maintained Trello cards
R. Batt EE	Some Flask implementation Hardcoded Search Page Redisigned Styling of Homepage (Login/Registration) Made all pages use same Bootstrap styling Allowed for multiple user accounts to be viewed
J. George EE	Hardcoded Profile Page Attempted to upload project onto Heroku Assisted with Cypher Query Design Adjusted website responsiveness
M. Cook CS	Hardcoded Homepage Recommended Logistic Changes to Topic Selection
S. Chawatama EE	Hardcoded Question Viewing Page Hardcoded Question Addition Page Contributed Basic Bootstrap Design Language
C. Meyer CS	Hardcoded attempt at Profile Page (Discarded - Not Compliant) Set up GITHUB Repository Set up Trello

Each member of the team was asked to write a piece on the page they hardcoded. These texts were adjusted and reformatted by P. Goos. The Relational vs Graph database section was written by S. Chawatama. The remainder of the report was compiled and written by P. Goos.

A.2 Project Leader Comments

For the sake of transparency I feel it would be best to discuss how I feel everyone's contributions were used and perceived. When calling meetings all members were keen to participate, however some would not always be prepared and would not have read over the Trello to keep up to date. All members of the team were requested to hardcode certain pages of the site. Hardcoded implies that only basic formatting and HTML should be done, no JavaScript or dynamic components further than those included with Bootstrap 4.1. A due date (on Monday) was set to complete the hard coded section, all but one page was received by that day with feedback being given on what changes should be made to the pages within a couple of hours. The page which was not received was the profile page which was being built by Mr George and Mr Meyer. It was brought to my attention later on that they were each completing parts of the page and then building on top of the work done by the other. The end result was Mr Meyer submitting a profile page that did not comply to almost all of the requirements set out for said page. Upon complaint to the two members Mr George asked for further clarification as he remade the page to meet the specifications. All other members' pages were received fast and were corrected in due time when requested to do so. As mentioned earlier in the report Mr Chawatama set the design language with his use of Bootstrap cards. The greatest problem within the team was the lack of feedback that I received from users. The Trello was continuously updated for the other members to see however I feel many underused this resource. Mr George attempted to set up Heroku, however unsuccessfully. Mr Batt and Mr George helped with smaller tasks and bug fixes along the route of me programming the site's python code. When asked Mr Cook and Mr Chawatama were always ready and offering help however this was not needed as often the problems were resolved soon after. The contributions of Mr Meyer do not equate to those given by all other team members, his report section submitted to me was written on the wrong page just to give an example. The lack of communication from him toward the other members of the group was unacceptable. One cannot simply avoid tasks assigned to them and then communicate a different understanding when the task is expected to be complete already. I hope this paragraph helps to gauge the team dynamic.

[illegible]

Pieter Goos

B Requirement compliance

Page	Comment	Req. No.
Hompage	The homepage doubles as both the registration form and login form	1, 2, 6
Search	Can search for questions and users	4
Question	Shows questions and answers. Allows for bookmarking and upvoting.	7, 8, 12
Add Question	This page solely adds new questions	7
Profile	This page has all the required lists and can edit one's profile. Detailed discussion in a prior part of this report.	3, 4, 5, 6, 9, 10, 11

The only two non-compliance's are the fact that Requirement 9/10 never does not show a result and a user does not have to assign at least one topic to a question. Other than this all the requirements are complied with.

References

[1] N. White. A microblog application powered by flask and neo4j. [Online]. Available: http://nicolewhite.github.io/neo4j-flask/

[2] T. Bootstrap. v4.1 documentation. [Online]. Available: https://getbootstrap.com/docs/4.1/

[3] T. Point. Flask - quick guide. [Online]. Available: https://www.tutorialspoint.com/flask/flask_-quick_guide.htm

[4] Flask. Flask (a python microframework). [Online]. Available: http://flask.pocoo.org/