Black Team Newsfast

Practicum

Igor Berdnikov
Garry Davitt
Neil Grogan
Kevin Kennedy
Larry O'Shea

School of Computer Science and Informatics

University College Dublin

20 August 2015

Table of Contents

1 User Scenario: The Characters

Our target users are people who are interested in technology, and want to keep up to date with latest technology news, but don't have time to browse multiple sources of news. Time is a problem for many people in modern life, in the tech sector this is particularly true, people are expected to work late, and they are also expected to be "always on", available to work. This applies to most types of jobs in the technology sector: Project Managers, Software Engineers, IT Engineers and other professionals. Technology projects normally come with tight deadlines, with the project engineers expected to put in whatever hours it takes to make it happen. People in technology are naturally interested in technology news, or even expected to keep up to date as part of their jobs.

News now comes from multiple sources, even for specialist news like technology. The Newsfast website can save time for these people, who want one source for their news. We personalise news for them, so the news is relevant to what they're interested in. Technology workers are our most important sector, other users we believe will be interested in the Newsfast web site include: academics, students and people like hobbyists who are interested in technology.

Our users are important because they tend to be highly educated, affluent and connected to the world at large. This makes them highly attractive to advertisers, which is important for our business model.

2 Technical Problem

Our system exists to personalise the news to the individual user. The current content of any major news website is heavily biased by editorial. We aim to specialise on the the technology sector to start, as we feel special purpose aggregators build more loyalty and should be more accurate. We aim to mix curation (of sources) and personalisation of news to the individual user. We will use our narrow focus to hone in on what drives our users and their needs. This will also serve as a proving ground to get the fundamentals of our product working to a high standard. After this, expansion of our core technology and concepts should be much, much easier.

The core problem is accuracy in tailoring the news to the users' needs. We believe a mix of a **ranking system**, **curation of feeds** and **cues from the user** will aid us in trying to solve this problem. We can demonstrate this by the fact that major aggregators still do not try to personalise the news for the user, or take an all algorithmic approach (Google News) or an all user curated approach (Reddit/Hacker News). We believe both methods have their advantages and disadvantages, which is why our method combines a mixture of both.

We can graphically illustrate the problem by asking why a died in the wool Android user has to skim articles about Apple's iPhone to get to the technology news that he or she is interested in ?

The closest competitor in the technology news aggregation space is Techmeme. Techmeme has no user personalisation, just a list of curated headlines to “suit” every user who visits the website and it makes you leave the site to view the article. Other websites of note include the Reddit /r/technology subreddit and Hacker News. The Reddit technology section is heavily moderated. We believe we are different from these competitors by offering personalisation to the user.

3 The Technical Solution

3.1 What does it do ?

The Newsfast system:

- Imports a large number of RSS feeds related to Technology news - this can be ‘curated’ on the Django Admin GUI
- When the user goes to the website, they are shown a landing page, from which they can skip sign in to see default news articles or sign in via their Twitter account
- When the user signs in, they can pick from a list of technology news topics
- The Technology news displayed on the website is then filtered or personalised according to the topics
- The articles are ranked according to a scoring system, the aim of which is to present the most interesting news to the user
- The system also learns user preferences, based on what articles they click on
- The system is designed so that it can be used for any type of news. For example, Fashion News could be shown by changing the RSS feeds and the Topics.

3.2 How does it work - the Front End

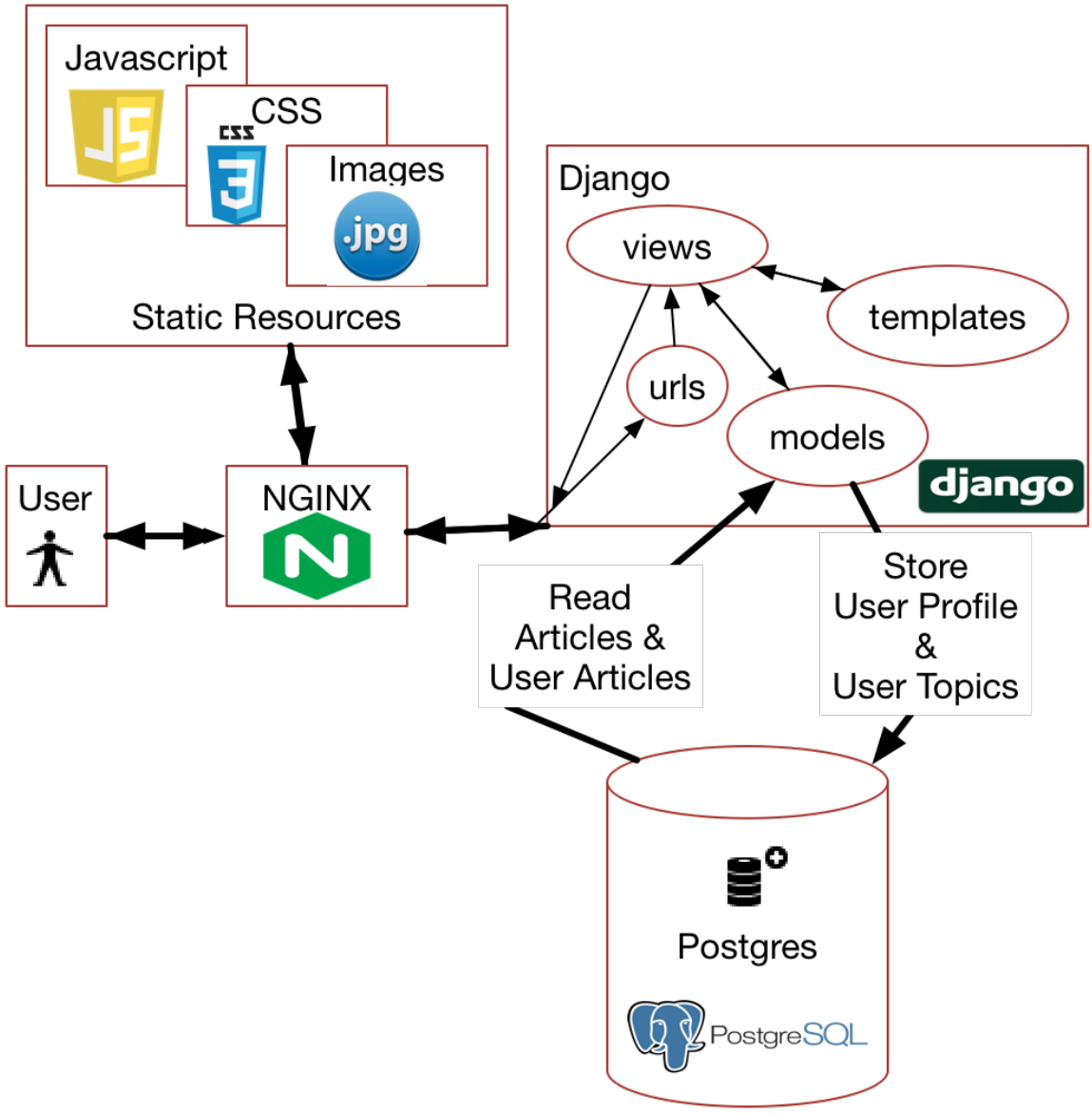

Figure 1 System Diagram for Front End

The diagram above shows the main system components in the Front End, the User, NGINX web server, the Django Web and Postgres. The UI resources of Javascript/jQuery, CSS3 and Images are shown.

User Login

When the user goes to the newsfast website for the first time, they have the option of signing in via Twitter. The user is prompted to grant access to the Newsfast application. If the user grants access, the application receives an access token from Twitter. This access token is stored as a cookie in the web session. The access tokens for each user are unique and allow us to get their twitter user id which is unique and doesn't change (Twitter allows username changes).

The sign in process is shown below:

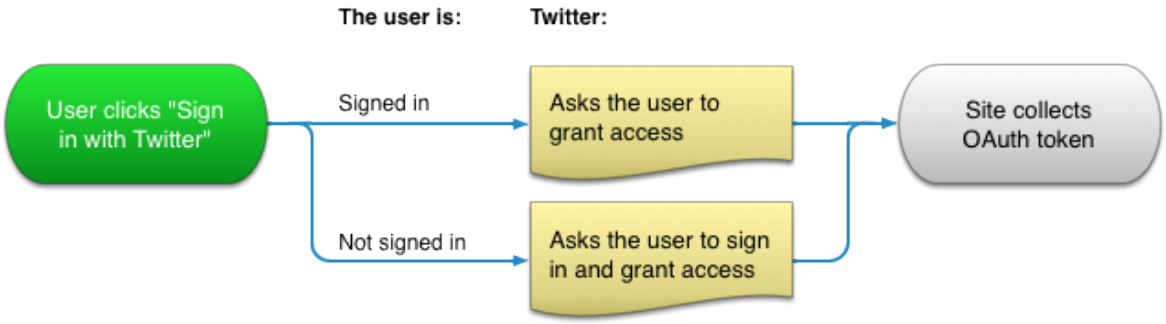

Figure 2 Sign in process with OAuth and Twitter

When the user login is complete, their Twitter Id and Twitter Screen name is stored in the Django UserProfile table. The Twitter Id is then used in our system to store other user data.

Figure 3 Newsfast Landing page, helps encourage users to sign in and tells them what the product offers

User Selects Topics

After Twitter sign in, the user can select tech topics they are interested in. The topics will be used to personalise the news for them. The user selected topics are stored in the `UserTopic` table.

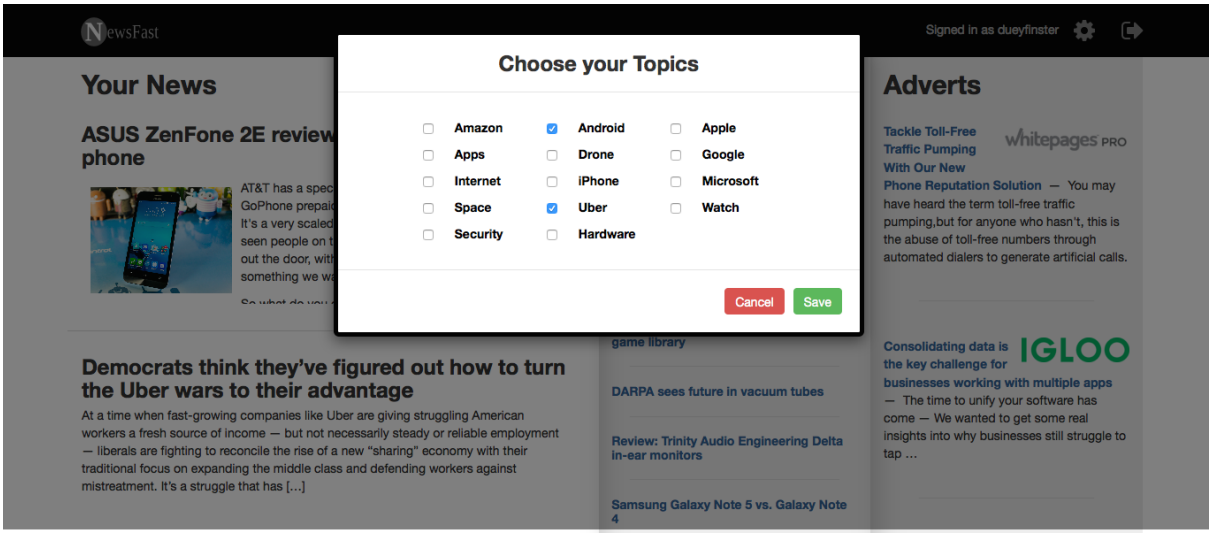

Figure 4 Choosing topics, the user gets plenty of choice

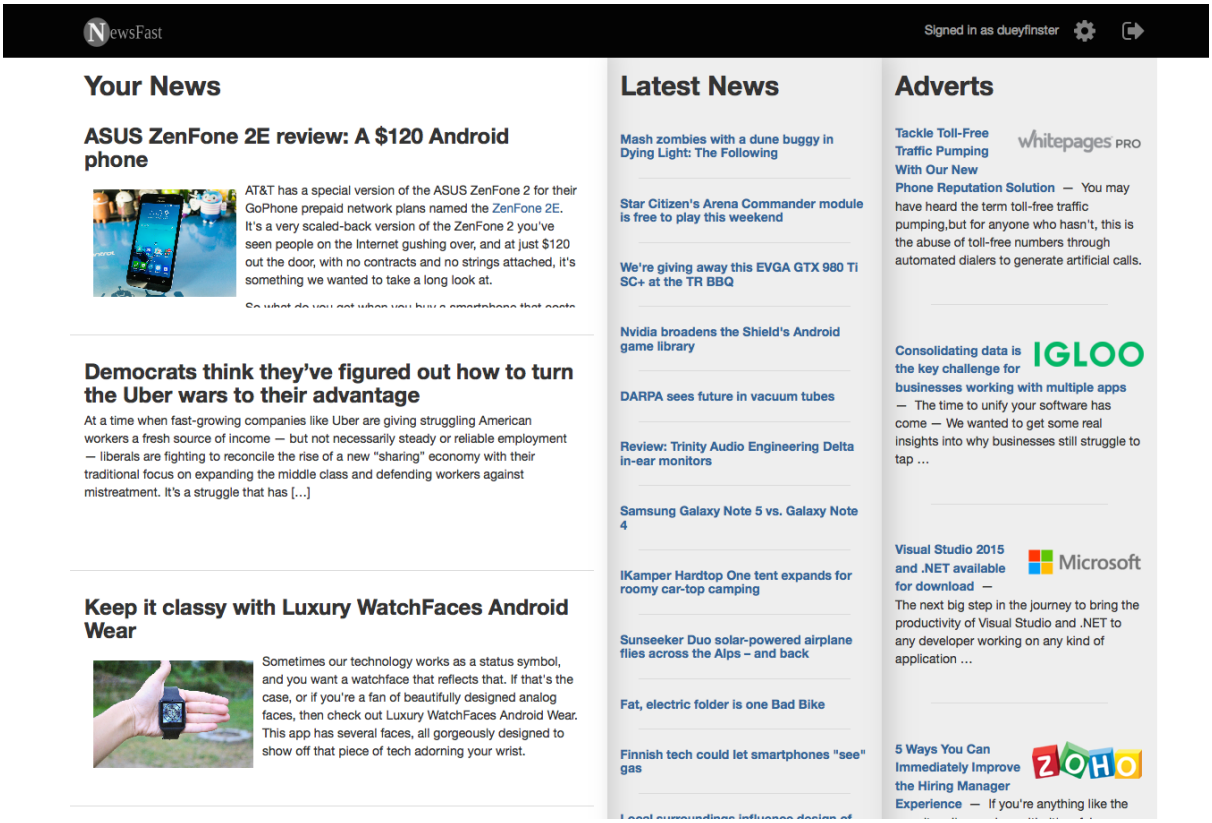

Figure 5 Main site after a user has chosen topics

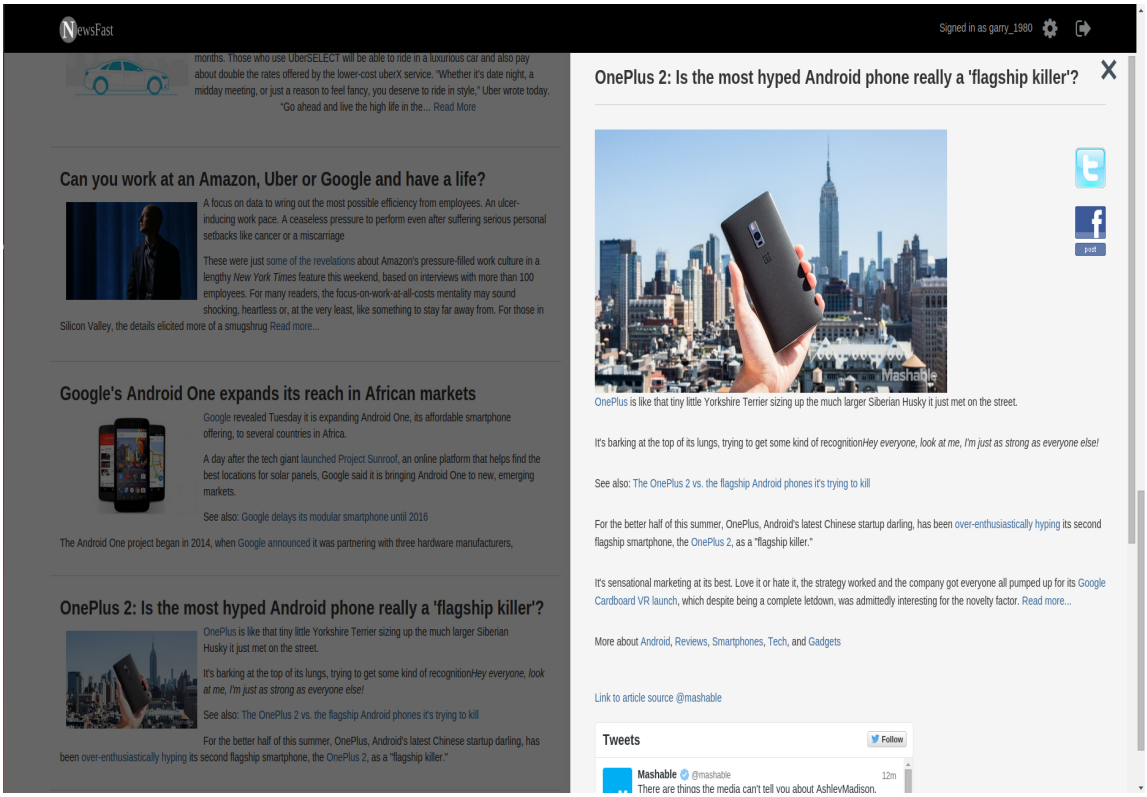

Figure 6 Main site showing overlay after user clicks on article

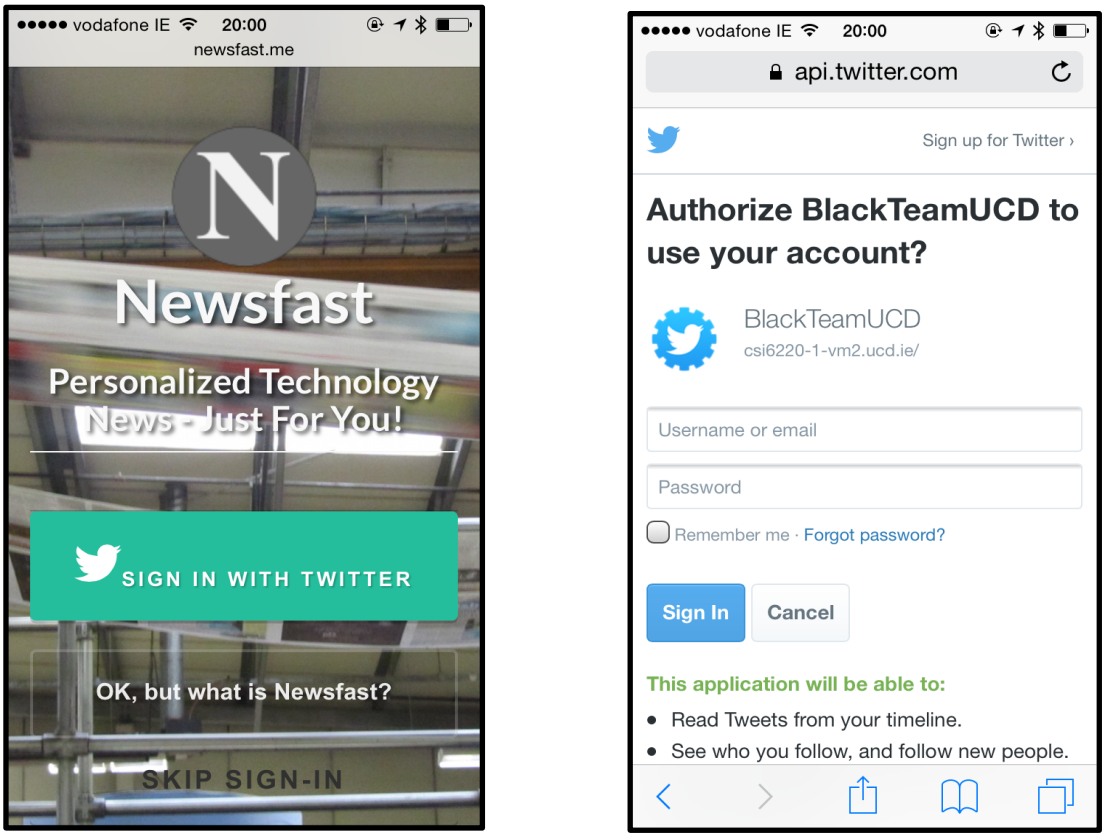

Figure 7 Landing page and Twitter sign in page on mobile

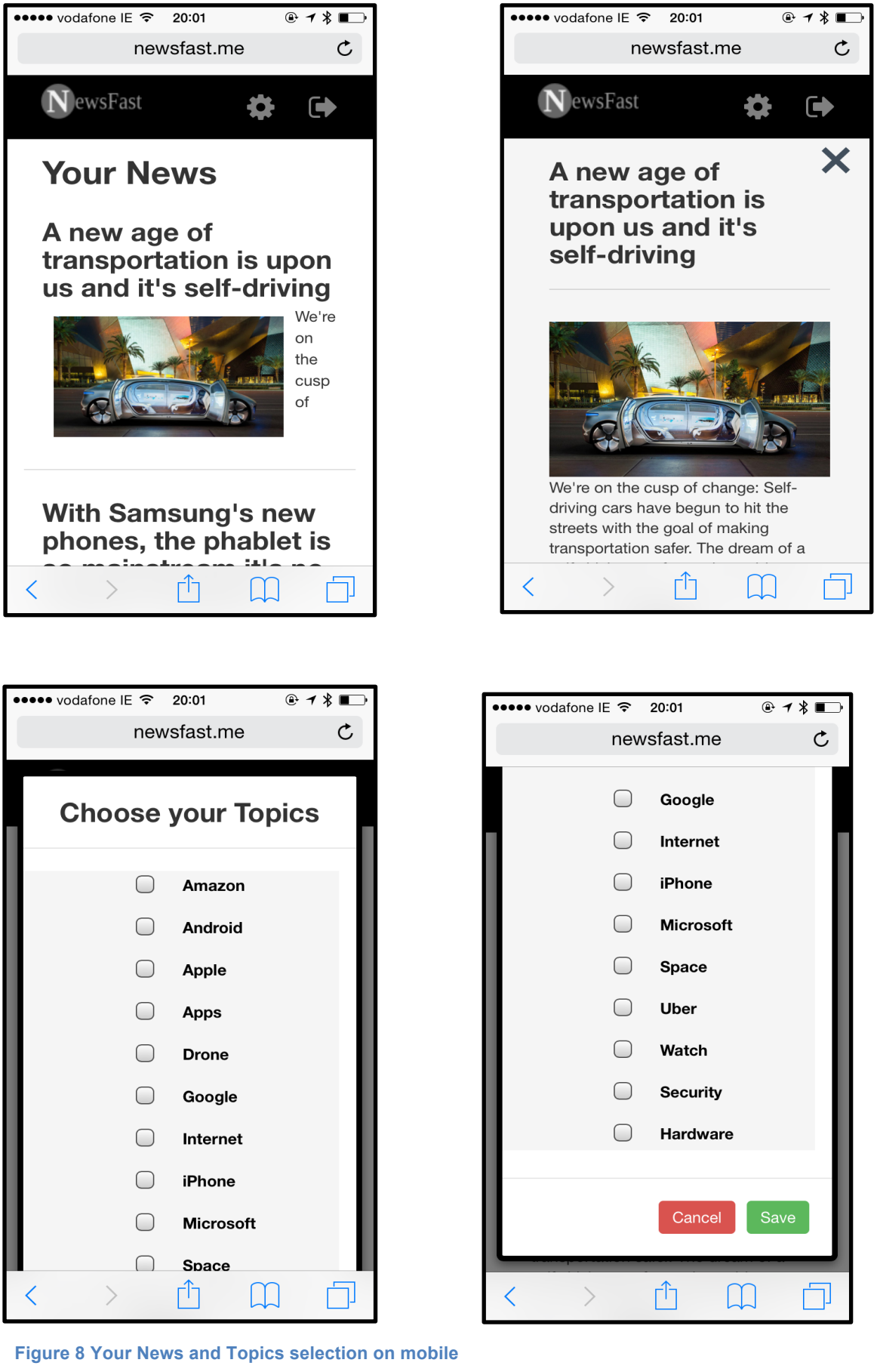

Figure 8 Your News and Topics selection on mobile

Front End - User Interface components

The user interface components are:

- Javascript
- jQuery
- CSS3
- Ajax
- Bootstrap

3.3 How does it work - the Back End

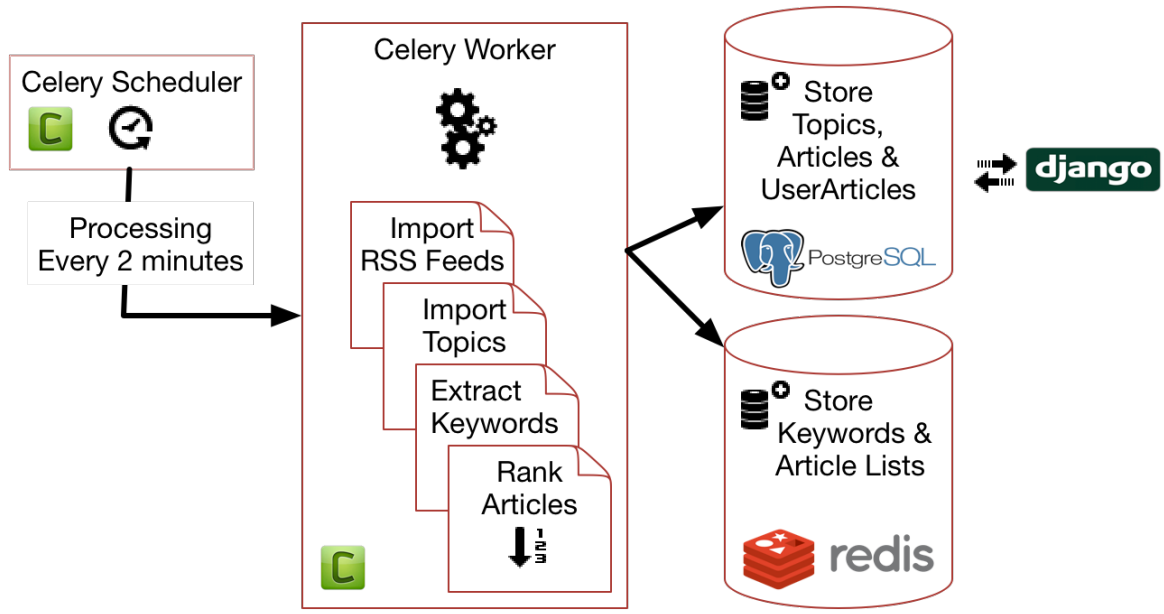

Figure 9 System Diagram Back End

System Startup

On startup the Celery Worker calls the following functions once only, this is to store important data into the postgres database where it can be edited on the django admin interface

- *import_feeds* - read the RSS feed url information from file and store in the database. There are about 50 RSS feeds.
- *import_topics* - read the global topic list and store in database
- *import_tech_keywords* - read the tech keyword list and store in database

Processing RSS Feeds

Every 2 minutes the Celery scheduler queues a task to process RSS feeds which is done by the Celery worker. Processing RSS feeds is the most intensive part of the system as a lot of data processing is done. The following steps are involved:

- Iterate through the articles in each feed
- Save the articles in the database in table `Article`
- For each user extract keywords from the article, according to user topics selected by the user
    - Keywords are stored in Redis for quick and easy access

- If the article matches the user topic preference, then store Article ID and Topic ID as per user in the database (table `UserArticle`) - this will be used to display Articles relevant to the user selected topics
- The user article is assigned a ranking based on a number of factors which are designed to ensure the most relevant articles are presented to the user

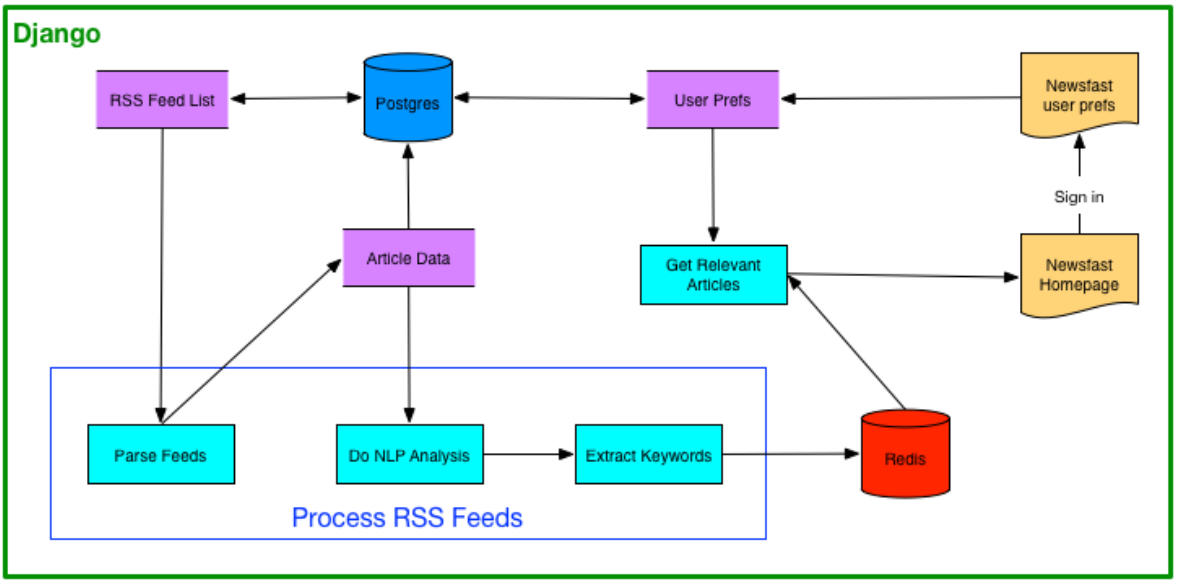

Figure 10 Logical Diagram for Backend

Extract Keywords

Keywords are extracted from each article and stored in Redis key-value store. The extraction is done using nltk python library and our own python code. Keywords are extracted from both article title and article body and stored separately.

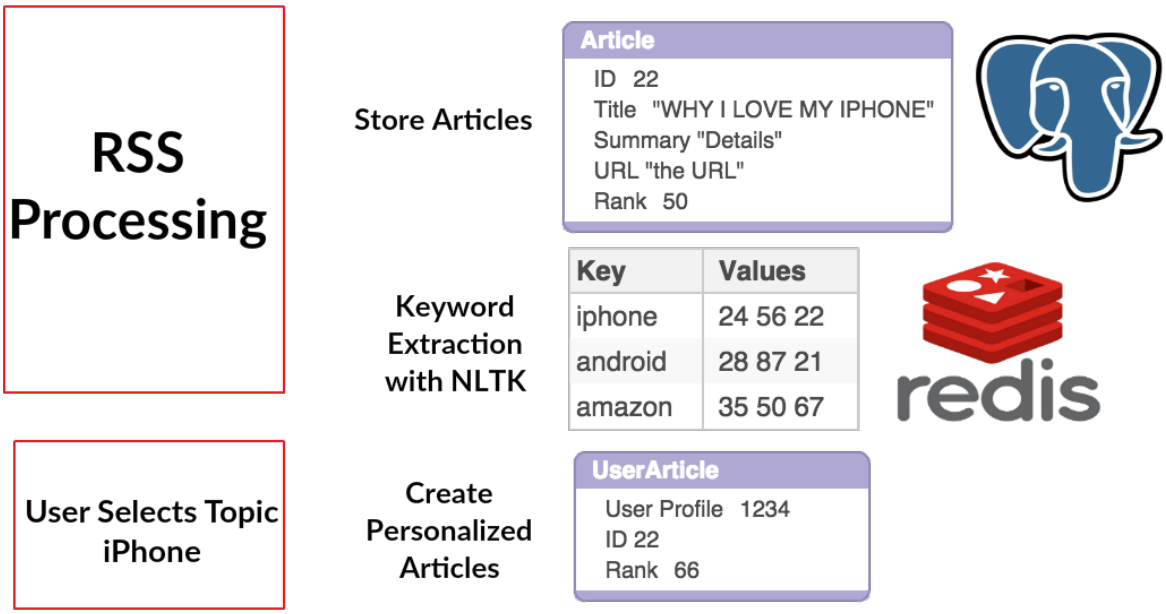

Figure 11 Diagram explaining Personalisation

Redis is perfectly suited to storing key-value lists, in our case the key is the Article keyword, and the list of values is the list of Article Ids. Every time the keyword extractions runs on an article, the article id will be added to each Redis list for the particular keyword.

When the User then logs in and selects a topic, a Redis lookup is done to get the Article Ids for that topic, we then store those article ids in the User Article table, with a rank for the article. The entries in the User Article table for that User, are the set of personalized articles for that user.

Ranking Articles

`UserArticles` are stored with a ranking value, the ranking value is the key criteria for displaying articles on the web page per user, so the ranking is key to personalisation for the user. The motivation is that if the User has selected topic iPhone, and there are 500 Articles on iPhone, we want to give the User the most interesting iPhone articles.
The ranking scheme for articles is shown below, note that it is and can be adjusted during on going testing and customer feedback.

Factor	Value	Comment
RSS Feed low quality	0	Judged manually, based on factors seen below
RSS Feed medium quality	+10	Judged manually, based on factors seen below
RSS Feed high quality	+20	Judged manually, based on factors seen below
Freshness (brand new)	+30	First time article is processed
Freshness >= 12hrs	-10	Every 12 hours, minus 10 to maximum of 32 hours (-30 points)
Source	+20	Optional, can add +20 bonus to better quality sources
Has Tech Keyword	+10	Multiplier, every tech keyword adds 10 points (no maximum for now)
Has User Keyword	+20	Multiplier, every tech keyword present for user adds 20 points (no maximum for now)
Has picture	+20	Once off score, may be adjusted to only work on sources with good quality pictures

Table 1 Ranking values for factors we considered important in Articles

Learning User Preferences

An important feature in the system is to learn user preferences in order to recommend articles that they would be interested in, based on what articles they have previously clicked on.

The key points to explain how this is done is as follows:

- When the user clicks on an article, an entry is added to the `TrainingSet` table, this contains User Id, Article ID, and Tech Keywords
- After some time there will be 100 `TrainingSet` entries per user
- When new articles in the rss feeds are processed, the new article is checked for similarity to the training set for the user, by counting how many keywords from the article also exist in the training set
- A rank is assigned based on the percentage of keywords from the new article that match the training set (using k-nearest neighbors algorithm)

Backend - Technical Components / Data sources

The backend technical components are

Technology	Reason used
Docker	for build and deployment
feedparser	used to parse RSS feeds
nginx	web server, reverse proxy to dynamic/static files
gunicorn	python dynamic web server
redis	used for easy and quick keystore access
requests	used for HTTP encoding / decoding
selenium	used for automated web testing
tweepy	used to get Twitter information
authomatic	used for Twitter sign in
celery	task queue
nltk	NLP processing
django-admin-bootstrapped	used for Django Admin look and feel

coverage, coveralls, pep8, pyflakes	used for code quality

Table 2 Backend components and why they are used

Newsfast ERD

The diagram below is automatically generated from django models and shows the main relations between django tables.

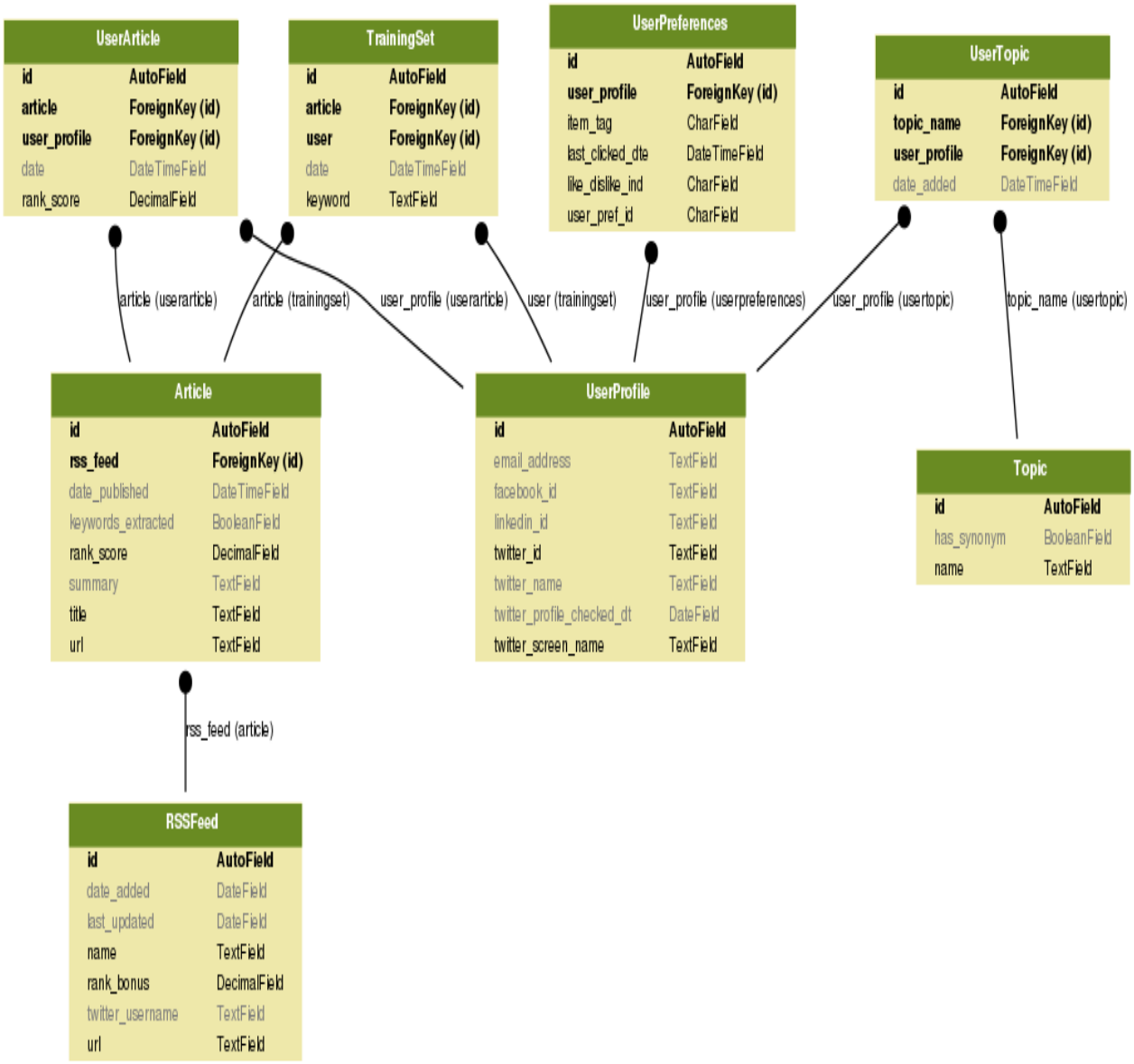

Table 3 ERD Diagram

The tables contain information as follows:

- `Topic` contains global topics entered by administrator
- **`UserProfile`** contains the twitter id and name. Any per user data will have a foreign key relationship to `UserProfile`
- Article contains the articles from the RSS feeds
- `UserTopic` contains the topics per user - it has foreign key of Topic and foreign key of `UserProfile`

- `UserArticle` contains the IDs of Articles per user - these are the personalised Article IDs which will be shown to the user. This table has foreign key of `Article` and foreign key of `UserProfile`
- `TrainingSet` contains ID of Article that User has clicked on. This is used for learning. This table has FK of Article and FK of User

3.4 Example input/output and interaction with different components

This is an example walk through of how the system works (the technical detail is described earlier in the document) :

- The system starts up, the Celery worker imports the list of RSS feeds and defined Topics into the database
- Every 2 minutes the Celery scheduler queues a task for the Celery worker to process RSS feeds
- The Celery worker processes RSS feeds, storing each article in the database table `Article`.
- The User goes to the newsfast website for the first time. The newest articles from Articles are displayed.
- The User signs in via Twitter, their Twitter details are stored in UserProfile table
- The User selects Tech Topics that they are interested in, these topics are saved in table Topic for the User
- Once the User saves topics, a function is called to get Articles matching these topics, the Article Ids and Topics are saved in `UserArticle` per user
- Articles in `UserArticle` are displayed to the User
- Any new Articles added from RSS feeds are added to `UserArticle` for the user, if the new Article contains keywords matching the Users topics
- As the user clicks on articles, the articles are added to the users training set, this is learning the preference of the User
- When new articles arrive, they are checked for similarity to the training set and added to the `UserArticle` table if they are similar, with a rank assigned based on percentage similarity

3.5 Software Development and Environment

Docker technology

We are building and deploying our system using Docker containers. They key benefits for us doing that are:

- Enables us to use continuous integration - code pushed to github is automatically pushed to the UCD VM
- Guarantees that we all use the same environment for local development and testing
- Software services are isolated in Docker containers - this reduces complexity, eases integration and also allows the containers to be scaled if needed. For example, to double our processing power we would just need to add a second Celery worker container

The Docker containers are as follows:

- NGINX
- Web - containing the main Django MVC system
- Postgres DB
- Redis Keystore
- Celery Scheduler
- Celery Worker (which can be scaled up or down by adding or removing workers)

Automated Testing

We added a number of automated tests using selenium, the automatic deployment to the UCD VM is dependent on the tests passing.

Continuous Integration

One of the benefits of Docker is that it enables us to use continuous integration, because the docker environment is the same on all our laptops and the virtual machines.

The development process is as follows:

- Develop on local Docker environment
- Push change to github
- Circle CI and Travis CI automated tests run
- If tests are successful, automatic deployment to one of the UCD VMs
- We receive notifications from github, Circle CI and Travis CI on our continuous integration channel in Slack

The full CI process is shown here:

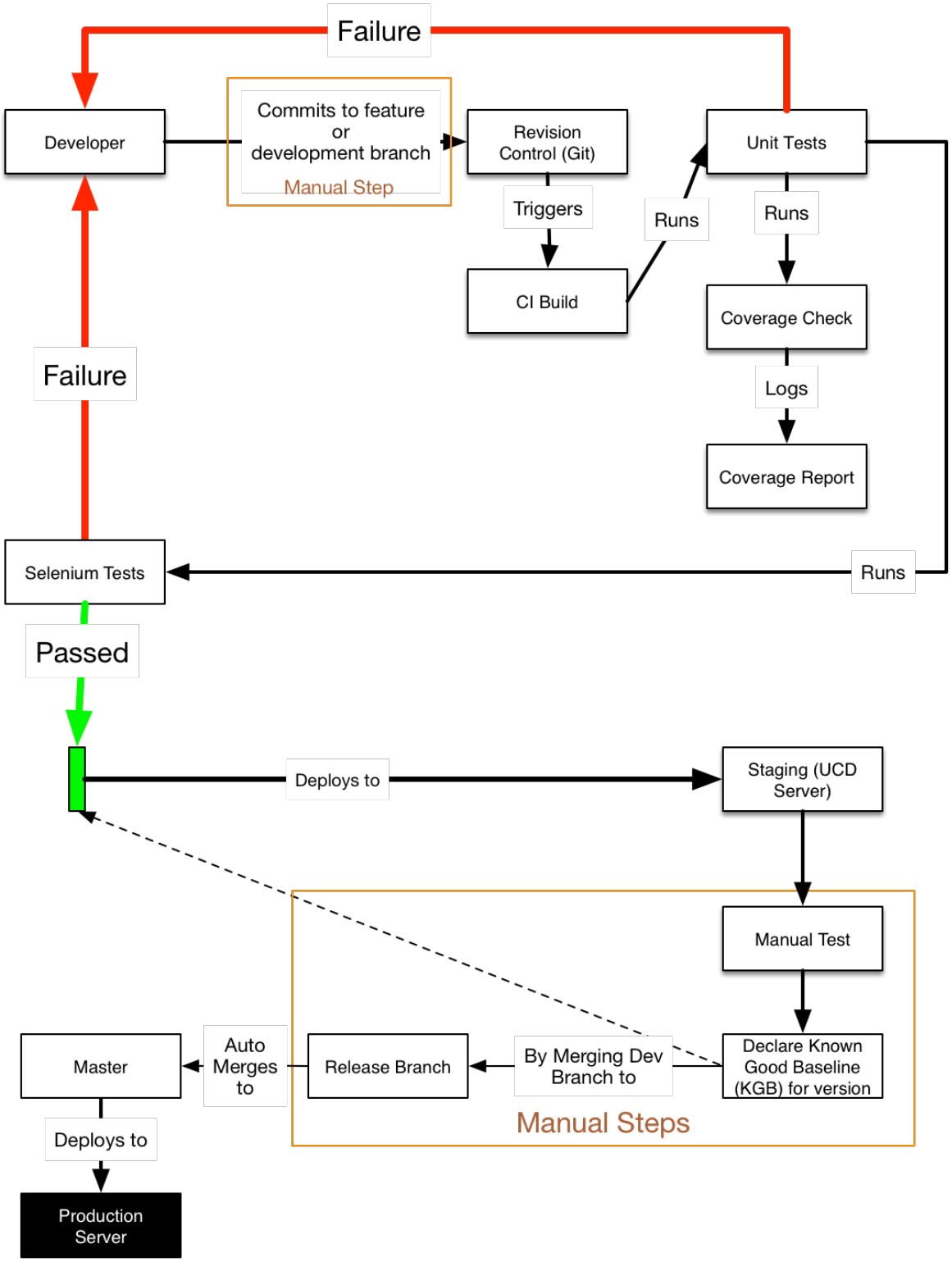

Figure 12 Our Continuous Integration flow

Example notification on slack is shown below:

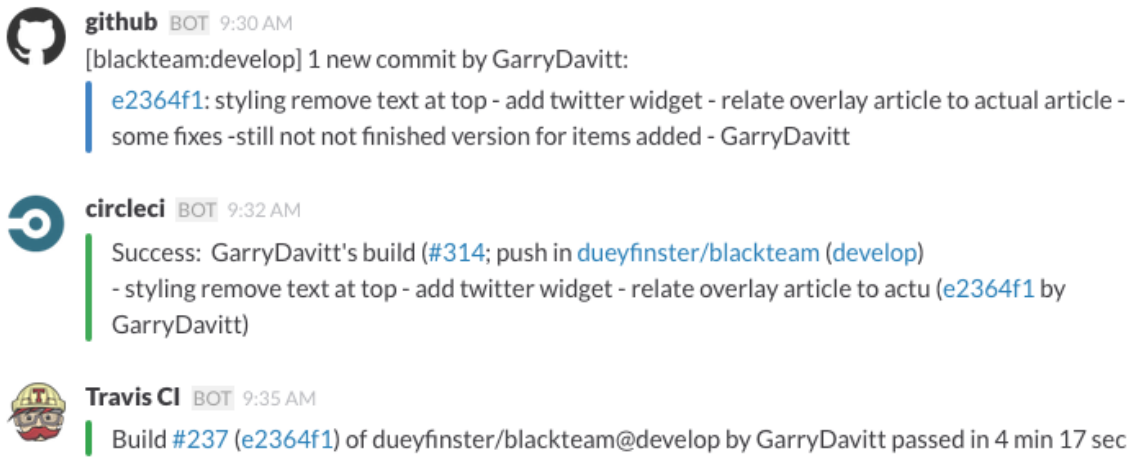

Figure 13 Example of notification of build, test and deployment robots to our Slack chat channel

4 Integration

We are also using Twitter to authenticate our users using OAuth 1.0a technology. This provides us with a fast and efficient method or registering users to our website and to authenticate them every time they want to use it. Crucially the user will have limited interaction with their computer. If they are already logged on to Twitter then they will only need to click on a button on our webpage and the registration process will run in the background.

Google Adsense will provide us with our source of income. During the initial stages we will be relying on this technology as our sole source of income as it's simple, fast and the content is managed by Google.

Our website interacts with and enhances the the RSS & OAuth technologies to provide news content that is personalised. It is this personalisation aspect of our technical solution that is our Unique Selling Point (USP). There are many websites already in existence that collate RSS feeds from different sources into one destination for users. Typically these sites offer little or no customisation and simply show the latest news to all users regardless of their interests. NewsFast will use OAuth (Twitter sign in for now) to keep track of the individual users. Initially the personalisation will be based on a list of topics that the users has selected as being relevant to them.

The main risk for the website, is that we are relying on free content from the RSS providers. There is no reason why they should provide this content to a successful website, especially if the website is commercially successful based on their content. If the website was very successful the RSS provider could remove the RSS feed, or even just blacklist the newsfast website IP and continue to provide the RSS feed to non-commercial sites.

Due to time constraints with this project we have not been able to go to each RSS provider and ask them for permission to display the RSS feed from their site. We are legally obliged to do this due to copyright laws. This would certainly become an issue if Newsfast became commercially successful.

5 The Impact

Our system will be impactful in the fact that it blends curation and algorithmic processing to present the user with the best news for them. No other service seems to have catered to this market successfully. How we plan to do this is by aggregating the most successful news feeds/competitors and using them as feedback for our algorithmic choices. We see many competitors in the market who just do aggregation (Techmeme) or user-submitted aggregation (Reddit), but none have got personalisation done right. Arguably, Facebook currently has the best personalization (Facebook Newsfeed) and they are now trying to extend this to news (Instant Articles). But it's still a walled garden of outlets that have opted in with Facebook and not the best of the open web, which is what we harness.

We also think it is the correct approach to start in a small subsection of the news and expand the service later to cover more topics of interest. The best sources of really valuable content tend to serve their niches (Financial Times, The Economist) really well and then expand beyond that once they have a trusting and loyal audience. In this way, the impact, at least initially, will be a slow build. After all, the Economist went from a regional London title to a globe spanning operation over the span of about 100 years, surely we can replicate this in the digital age in a much shorter timeframe!

Newsfast will start having an impact once we can get appropriate scale to test and fortify our algorithms and feed choices. We expect to be able to grow fast if we can convince a core group of influencers in the media space to reference us as a news source. We can also aim to have a big social media presence, being referenced with the "via" annotation. People like to share what they feel is interesting and relevant to them, so it is by presenting them with this content we can start to get ripple effects of being shared across social media.

We also feel although mobile applications get the most buzz and attention, actually the web is a better place to be able to iterate quickly. Research by Flurry Analytics shows that users will immediately uninstall a mobile application they do not like[1]. However, a website like ours, the user has no such ties to us and we can easily test retention and other key metrics, some of which are harder to record on mobile devices, like uninstalls (for obvious reasons). Flipboard is a hugely popular aggregator on mobile, which also offers limited personalisation, but grew its audience by 75% by broadening its offering and reach to a more traditional website[2]. In summation, we feel we're well placed to make an impact with scale and focusing on quality of personalisation.

---

[1] "Why People Uninstall Apps" Forbes. Web. 13 Aug. 2015. <http://www.forbes.com/sites/ciocentral/2013/11/21/why-people-uninstall-apps/>

[2] "Flipboard now has 70M monthly active users, says CEO Mike McCue" Venture Beat. Web.

[2] "Flipboard now has 70M monthly active users, says CEO Mike McCue" Venture Beat. Web. 13 Aug. 2015. <http://venturebeat.com/2015/07/14/flipboard-now-has-70m-monthly-active-users-says-ceo-mike-mccue/>

6 Reflections

6.1 How successful is the solution ?

We believe that our product is simple to understand and to use, and that we've executed it well. The user interface of our website now looks professional on desktop and Mobile. Our personalisation is not complicated, but it's probably the one thing that not a single user has complained about. Our user survey initially had negative comments, but those became more positive as time went on. We received 22/27 positive statements on personalisation, and 24/31 positive statements about using the product again. We do note that the feedback came from work colleagues. In terms of the product being commercially successful, we recognise that attracting users is very difficult to do, and there are clear difficulties with copyright, and in general that we don't own the content.

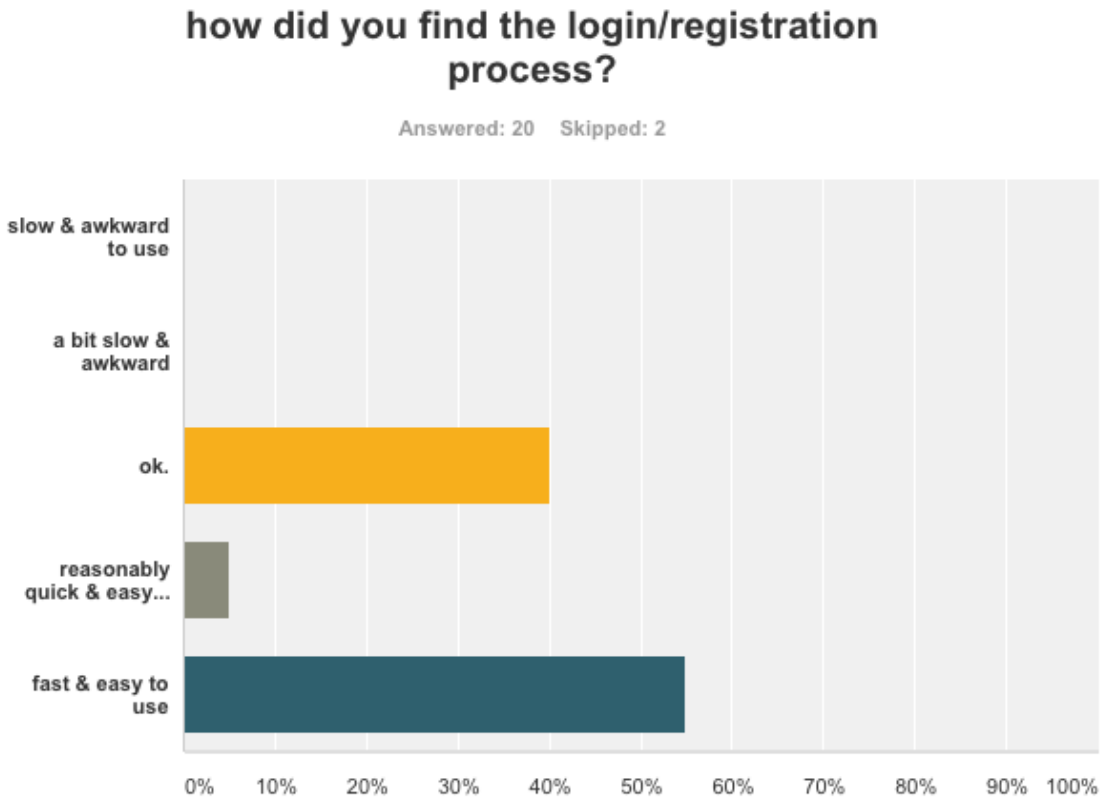

Figure 14 SurveyMonkey result of how easy users found sign in process

6.2 How appropriate were the choices you made about technologies to use ?

The choice of Docker to enable continuous integration, and ensuring we all have the same environment has been very successful. It has meant that we've never spent any time integrating work as the local environment is the exact same as the deployed environment. It

has been more painful to use for Windows users, and that has been frustrating at times, but it has certainly saved the group a lot of time.

Docker has also made some of the other technical choices easier, for example integrating Celery and Redis was straightforward as there were Docker images already available for these services. Redis was a good choice for storing keywords and article IDs, as it's well suited to storing data types like lists, dictionaries and sets. Using Celery for task queuing has worked very well, it means the intensive processing happens on the Celery worker container and not the Django web container - hence the website remains responsive for the user.

Celery also gives us a good story for scaling the system, we can add Celery worker containers or move the Celery processing to a different virtual machine quite easily.
On the user interface we choose Javascript, JQuery and Bootstrap. They are good, safe choices as those technologies are used in a large number of commercial websites. The python modules we've chosen have meant we're not 're-inventing the wheel'.

6.3 What were the biggest challenges you faced as a team ?

Time is a constant issue, it's difficult to juggle jobs and families with the project. Different people will have different time pressures which will then change. Another challenge is location, because the team is spread out with some people in Dublin, some in Athlone/Galway, some travelling in Wales. We've dealt with that very well using Slack chat for communication and Google hangout calls for the whole group at least three times a week. The group interaction has been very high despite not being able to physically meet up, so location hasn't been a problem. Technically it's worked out well, as we've tended to specialize as time went on, for example between back end, front end, and documentation deliverables. We've also switched around these roles as required, which keeps it interesting for us.

6.4 How effective were the project management and software development methodologies ?

We've used Scrum and Trello which has been quite effective. We split the time into sprints at the start and we have been good at updating Trello. It was challenging at the start of the project, where there was sometimes a conflict between agile "let's get it working quickly" to "let's agree a document on how it should work". We found a good middle ground by having online session where we drew on a whiteboard and then made decisions about the structure of the project very quickly.

6.5 What lessons have you learnt - what problems are hard/easy ?

One of the biggest challenges early on was scheduling. As we are a part time team, we have to constantly work around other commitments. This meant every team member had to be on the same page with regards to when we needed to hit our targets to make our minimum viable product possible. We also encountered very different working styles, some team members were highly collaborative, whereas others preferred to do a chunk of work in isolation and submit the work upon completion.

Another was agreeing how the user interface should look. We pretty much stuck to our initial mock ups, but that still left a lot of small choices along the way we didn't anticipate. Sometimes the discussion was heated, which was great as it showed the passion of the team members and their attachment to getting the product to a high standard. We learned to prioritise different aspects, and practice give and take in our opinions among the team of the finer points of the product.

We didn't prioritise getting user feedback at the start because of the lack of a minimum viable product. It's always lower down the priorities then actually trying to build the product. But the early feedback we learned is vital. Some members of the team sent our survey out very early, which there was disagreement on, as the product was very early on in the lifecycle. It was felt embarrassment may ensue, as colleagues would exclaim "Is that what you're working on?". In fact we actually found the opposite, people were quite supportive but critical. This helped immensely as we could survey the same people, who would see how we responded to their initial feedback, and ultimately gave us validation about the core mission of the product.

In terms of the team skills we did not have much User Interface (UI) experience so we had to have a ramp up period, this was not ideal. The biggest challenges in terms of the UI was to keep everything on the one page so the user did not have to be switching around between multiple web pages. The popup for user preferences and the overlay for the article content solved this. Once we had the site up and running we realised from the survey that a lot of users were trying to access it through mobile. Between this and supporting two browsers (Firefox and Chrome), it was quite tricky to make sure the site was working well on desktop and mobile phones.

In terms of skills and knowledge, we've learnt a lot over the course of the project:

- Django and Python programming
- UI skills as mentioned - JQuery and Javascript
- Docker technology
- Business Models - this area was new to most of us

We all derived a lot of satisfaction from being able to deliver a complete product, with back end, front end, business plan and getting feedback from users. In our day jobs we work on one small part of a product, producing a product from scratch is very satisfying.

7 Bibliography

2008. Web. 13 Aug. 2015. http://www.smashingmagazine.com/2008/01/10-principles-of effective-web-design/

"120 Useful CSS3 Tutorials Examples And Tricks for Designer." *Fresh Design Web RSS*. N.p., Mar. 2015. Web. 13 Aug. 2015. https://www.freshdesignweb.com/css3-tutorials-examples/

Alchin, Marty. *Pro Django*. Berkeley, CA: Apress, 2009. Print.

"CSS Overlay Techniques." *Codrops CSS Overlay Techniques Comments*. N.p., Jul. 2013. Web. 13 Aug. 2015. <http://tympanus.net/codrops/2013/11/07/css-overlay-techniques/>

"CSS3." *CSS3 Info RSS*. Web. 13 Aug. 2015. <http://www.css3.info/>

"CSS3 Introduction." *CSS3 Introduction*. Web. 13 Aug. 2015. <http://www.w3schools.com/css/css3_intro.asp>

"Django Development With Docker Compose And Machine." *Django Development with Docker Compose and Machine*. Web. 12 Aug. 2015. <https://realpython.com/blog/python/django-development-with-docker-compose-and-machine/>

"Embedded Timelines." *Embedded Timelines*. Web. 13 Aug. 2015. <https://dev.twitter.com/web/embedded-timelines>

"HTML(5) Tutorial." *HTML Tutorial*. Web. 13 Aug. 2015. <http://www.w3schools.com/html/>

"How To Create an Overlay in Css." *CSS Overlay Techniques,* Web. 13 Aug. 2015. <http://www.corelangs.com/css/box/overlay.html>

"Introduction To Celery¶." *Introduction to Celery — Celery 3.1.18 documentation*. Web. 12 Aug. 2015. <http://celery.readthedocs.org/en/latest/getting-started/introduction.html>

"JavaScript." *Codecademy*. Web. 13 Aug. 2015. <https://www.codecademy.com/tracks/javascript>

"JavaScript Tutorial." *JavaScript Tutorial*. Web. 13 Aug. 2015. <http://www.w3schools.com/js/>

"Key Concepts." *Knockout : Home*. Web. 13 Aug. 2015. <http://knockoutjs.com/>

"Welcome To the Docker User Guide." *The Docker user guide*. Web. 12 Aug. 2015. <https://docs.docker.com/userguide/>

"JQuery." *jQuery*. Web. 13 Aug. 2015. https://jquery.com/

Mitchell,T Machine Learning, McGraw-Hill Science, 1997