A configurable social network for running IRB-approved experiments

Mihovil Mandic
Mihovil.Mandic.21@Dartmouth.edu

Follow this and additional works at: https://digitalcommons.dartmouth.edu/senior_theses

Part of the Databases and Information Systems Commons, Data Science Commons, Models and Methods Commons, Other Computer Sciences Commons, Science and Technology Studies Commons, Social Psychology and Interaction Commons, and the Software Engineering Commons

Recommended Citation
https://digitalcommons.dartmouth.edu/senior_theses/213

This Thesis (Undergraduate) is brought to you for free and open access by the Theses and Dissertations at Dartmouth Digital Commons. It has been accepted for inclusion in Dartmouth College Undergraduate Theses by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.
A configurable social network for running IRB-approved experiments

Mihovil Mandic

Undergraduate Computer Science Thesis

Dartmouth College

Advisors

Professor V.S. Subrahmanian

Professor Soroush Vosoughi

Professor Benjamin Valentino

In partial fulfillment of the requirements for the degree of

High Honors Bachelor of Arts

June 1, 2021
Abstract

Our world has never been more connected, and the size of the social media landscape draws a great deal of attention from academia. However, social networks are also a growing challenge for the Institutional Review Boards concerned with the subjects’ privacy. These networks contain a monumental variety of personal information of almost 4 billion people, allow for precise social profiling, and serve as a primary news source for many users. They are perfect environments for influence operations that are becoming difficult to defend against. Motivated to study online social influence via IRB-approved experiments, we designed and implemented a flexible, scalable, and configurable social network called DartPost. DartPost allows us to run highly customized social network simulations and easily obtain subjects’ consent for collecting data typically exclusive to Big Tech engineers. This opens up a wide range of avenues for further research, including analysis of highly important impression data, often missing in other social influence studies.
# DartPost Design and Implementation

2.1 Back-end ................................................................. 6
   2.1.1 Environment configuration .................................. 7
   2.1.2 Database ............................................................ 7
   2.1.3 Models ................................................................. 9
      2.1.3.1 User ................................................................. 9
      2.1.3.2 FollowerFollowing ........................................... 10
      2.1.3.3 ObserverObserving .......................................... 10
      2.1.3.4 Post ............................................................... 10
      2.1.3.5 Reposts .......................................................... 11
      2.1.3.6 Like ................................................................. 12
      2.1.3.7 Comment .......................................................... 13
      2.1.3.8 ReportUser and ReportPost ................................. 13
      2.1.3.9 SearchQueries ................................................ 13
      2.1.3.10 PostViews ..................................................... 13
      2.1.3.11 Messages and GlobalMessages ............................ 14
   2.1.4 Controllers ........................................................ 14
      2.1.4.1 Users ............................................................ 14
      2.1.4.2 Posts ............................................................ 15
      2.1.4.3 Like ............................................................... 17
      2.1.4.4 Comment .......................................................... 17
      2.1.4.5 Reports ............................................................ 17
      2.1.4.6 Search ............................................................ 18
## CONTENTS

2.1.4.7 Messages ........................................ 18  
2.1.5 Migrations ........................................ 18  
2.1.6 Data Export and Integrity .......................... 18  
2.1.7 Data Import and Seeding .......................... 19  
2.2 Front-end ............................................ 20  
  2.2.1 User Profiles .................................... 20  
  2.2.2 Posts ............................................. 22  
  2.2.3 Reports .......................................... 25  
  2.2.4 Search ........................................... 27  
2.3 Metadata ............................................. 29  

3 Use Case ............................................. 33  

4 Conclusions .......................................... 40  

References ............................................. 42
Chapter 1

Introduction

Since 2005, the Pew Research Center has been tracking social media adoption rates in the United States. Back then, only 5% of Americans used at least one platform, while today, that number has risen to 72% [36]. Counting globally, it is estimated [45] that at least 4 billion people log on to their favorite social platform once a month. Social media now surpasses print newspapers as a news source for Americans, and spending on digital advertising exceeds spending on all other forms of advertising combined. What makes these platforms so attractive is their ability to create and share multimedia with the entire world through an environment that helps these interactions feel more natural.

Given these facts, it is no surprise that the use of social networks as research tools is surging. As our world becomes more connected, the social network user base is becoming more representative of the broader population, which provides many unique benefits to researchers. First, the textual and multimedia content on social media tends to be very temporal, meaning that it is often easy to make a connection between the posts and some recent events. Second, social networks provide us a window into the daily life of many users around the globe, leading to an abundance of natural behavioral data. Third, these networks have precise targeting tools which researchers can use to focus on populations that are typically underserved or out of reach with traditional research tools. Finally, the costs of conducting such research are often low, as access to targeting tools, APIs, and preprocessed datasets provided by these networks is often free [28].

The rapid growth of social networks and the depth of their research tools is a growing challenge for many Institutional Review Boards (IRBs). Moreno et al. did a review of the ethics of social media research in 2013. Little had changed from then when they wrote
that “As of yet, there is little-to-no guidance from federal regulations or institutions, and very little existing literature, on how an IRB should review research protocols involving social media websites.” The risk of conducting such research and its potential impact on subjects’ privacy is difficult to comprehend sometimes. Still, one of the most challenging things of all is acquiring their consent to participate in the first place.

In their review, Moreno et al. [28] focus on three types of research on social networks that typically require IRB review: observational, interactive, and survey/interview. They consider the relevant risks and discuss the IRB review exemption criteria for each. When it comes to observational research, it is typically exempt from IRB review if the data being used is public, and the subjects cannot be identified in any way directly or through the identifiers available in the data. With interactive research, the researchers often need access to information that is not publicly available, and this requires interacting with the participants. Such interaction could start with a Facebook (or Twitter) friend (follow) request, which could lead to an unpleasant misrepresentation of the researcher’s intentions. Lastly, with survey/interview research, it is hard to obtain a face-to-face contact with the participants as well as parental consent if they are a minor. This might reduce the “opportunities for the researchers to observe participant reactions to the consent process” [28] and would make it challenging to explain every single part of the data that is being collected.

Another type of research is increasingly more relevant and can be interpreted as a combination of the three aforementioned. For instance, online social influence (OSI) operations experiments rely on expression data from social media platforms, which is public data describing users’ engagement with content. Then, researchers would combine this observational data with the survey data on what posts or users might have influenced their opinion on a specific topic. Lastly, they could choose to interact and directly communicate with the participants in order to run their social influence operation.

On Facebook, Twitter, and Instagram, engagement data corresponds to likes, reposts, comments, and follower relationships. A lot of social media content and activity is being designed to influence users’ behaviors and beliefs, and reports of OSI operations are now widespread. “Good” influence operations include campaigns focused on wellness, such as the Healthy Together Victoria campaign in Australia [47] and the state of Maryland’s social media campaign to promote mask usage [6].
Malicious influence operations include allegations of foreign state-backed influence operations in the 2016 US election [13], anti-vaccination campaigns [20], and COVID-19 related misinformation [35]. Therefore, understanding how influence occurs on social media is of utmost importance for almost every aspect of modern society. This has led to an explosion of research on online social influence from computer science [22, 29, 38], physics [25, 9], and the social sciences [4, 7].

It used to be challenging to tell a vast number of involved users on Facebook that their engagement data is being collected by someone who is not Facebook. This has changed as social networks introduced data collection through third-party apps which have to ask for user permission to get specific parts of their data. However, when it comes to experiments regarding OSI operations, the studies that use such apps typically have no way to determine the ground truth data about exactly whose opinions or behaviors (other than expressions) were influenced and to what extent. Notably, data about exposure to content, also known as impression data, is also missing in these studies, as acquiring that data can sometimes violate terms of use of social networks and raise certain ethical issues. These types of studies require positive control of the OSI operation to fully study their strategies, effects, and goals, which is difficult to do through third-party apps on Facebook, Twitter, or Instagram.

Facebook’s scandal with Cambridge Analytica was uncovered in 2018 and had since “sparked the great privacy awakening ” [23]. Briefly, Cambridge Analytica used a third-party app to access the data of several hundred thousand Facebook users. However, Facebook also allowed this app to access the data of the users’ Facebook friends, who did not provide any form of consent. Cambridge Analytica harvested data from millions of users through this network effect and misused it for political advertising. The scandal is now raising important conversations on user privacy, changing the way researchers access this data. Combined with the tight IRB guidelines and the terms of use of said social networks, these changes could limit the results of “good” academic research.

As a part of my reading course for this thesis, I interviewed Dr. Rand Waltzman, the founding board member at the Information Professionals Association. Dr. Waltzman is an expert in social media artificial intelligence research and has 35 years of experience in the information environment [30]. He pointed out that “good” IRB-approved experiments suffer from a strong asymmetric advantage.
“Bad” online social influence campaigns typically use sock puppet accounts, online identities specifically created for deception. With these accounts, they can infiltrate a large group of users and share content to influence their opinion. It all starts with a simple Facebook friend request or a Twitter follow, which is often loosely accepted, as “the absolute number of Facebook friends is often considered a marker of positive social capital” [28]. After forming a “friendship”, they engage with users and try to directly influence their opinion through various strategies. On the other hand, similar “good” IRB-approved experiments have to use third-party apps that instruct the subjects on what will happen with their data as the app explicitly asks for their consent. As these limitations directly impact the quality of research on potential defense mechanisms, defending against “bad” OSI campaigns is becoming increasingly challenging.
Chapter 2

DartPost Design and Implementation

Motivated to overcome these limitations, Prof. Subrahmanian, Prof. Vosoughi, Prof. Valentino, and I decided to create DartPost, an original, web-based micro-blogging platform that captures much of Twitter’s and Facebook’s functionality. I have solely been developing DartPost from scratch since April 2019, and it involved writing 34,000 lines of code. I intended to make it highly abstract, scalable, and customizable. Our goal is to have access to data, algorithms, and insights typically available to Facebook’s and Twitter’s in-house engineers but not included in their public APIs. This includes impression data, which consists of posts that users viewed at specific times and on certain pages. This way, DartPost can support all sorts of social media research, including, but not limited to, online social influence (OSI) experiments with thousands of recruited users.

One of the many advantages of DartPost is that it can easily be deployed many times. This allows us to run multiple concurrent experiments, where each can have its own isolated DartPost environment. One environment, or platform, is comprised of one front-end and one back-end web application, individually built using novel programming tools taught in COSC052, an undergraduate class on Full-Stack Web Development taught by Tim Tregubov [46].

Simply put, the back-end deals with everything but the user and admin interface. It contains all the logic when it comes to posts, user accounts, and access control. On the other hand, the front-end is the only thing that users or research subjects can see. It is a static web application user interface that communicates with the back-end through HTTP
2.1 Back-end

requests. This is commonly called a model-view-controller software design pattern, where the back-end handles the model-controller part, while the front-end handles the views.

The figure 2.1 shows the DartPost homepage that a user sees after logging in. Similar to most social networks, this page shows them the posts of the users they follow in reverse-chronological order.

![DartPost homepage – User’s timeline](image)

**Figure 2.1** DartPost homepage – User’s timeline

### 2.1 Back-end

The back-end application is called *dartmouth-socialplatform-api* [27]. It is maintained in a private eponymous GitHub repository which is instructed to automatically deploy all Master branch commits to *Heroku* [19]. The application was built using the *Express.js* [14], an open-source *Node.js* [31] web application framework. *Express.js* provides the server-side logic and communicates with our *PostgreSQL* [37] database using *Sequelize* [43], an object-relational mapper.

The *src* folder of the back-end application contains the *config, controllers, migrations, models, seeders*, and *services* source code folders. Each of these is described in its section below.
2.1 Back-end

2.1.1 Environment configuration

The config folder contains a config JavaScript file describing the .env variables used with Sequelize and the Express server. The file contains three JSON objects, one for each of the deployment environments: development, testing, and production.

The production configuration is set to look for a DATABASE_URL system environmental variable to connect to the database. The Heroku Postgres addon sets this .env variable to the Uniform Resource Identifier of the current production plan database, which contains the database name, port, username, password, and server URL. To perform better at scale, I defined a connection pool in this object so that the production server can reuse connections and not add additional overhead when establishing these repeatedly.

2.1.2 Database

Since I was building a generalized social network that should support many predefined relations, such as those between users and their posts, I decided to use an open-source SQL database called Postgres. Besides the common advantages of SQL over NoSQL for a relational schema, Postgres has some significant benefits when it comes to scalability on Heroku using their Heroku Postgres add-on. With some simple shell commands, I can migrate the database from a free to a paid performance plan every time we decide to run a larger experiment. This way, we can maintain a smaller staging database for free until we decide to upgrade it to a much larger, faster, and indexed database server for a short-term period.

In Figure 2.2, you can see the current database schema. It generally works for most experiments; however, it was slightly modified to accommodate the influence analysis experiment (Section 3).
Figure 2.2 General database schema and specific relations for the influence analysis paper. The big green arrows connect tables that have a relationship. Foreign keys are indicated by a blue arrow on the right side of the field description.
2.1.3 Models

Every model has a \textit{createdAt} and \textit{updatedAt} UNIX timestamp field. This way, we can track timestamp changes across different tables and look for temporal relationships in the data. For instance, if we receive three reports against a suspect user for acting as a fake account, we can choose to display a red “suspicious” label on the suspect’s posts. Then, we note the time when we started displaying the label and can then analyze how it affects the number of future reports against this suspect. If we incentivize users to search and report fake accounts, more users might choose to report them if they are already suspicious. From this point, I will capitalize the model names when used in database context for clarity, as some were named simply by what they represent.

2.1.3.1 User

At its core, the User model consists of a primary key (\textit{id}), \textit{email}, \textit{username}, and a hashed \textit{password}. In case a user wants to edit their profile, they can upload an image which in turn updates their \textit{profilePic} field with a URL, or change their \textit{firstName} and \textit{lastName}, which they can display above their Posts.

To separate User roles, I decided to use simple Boolean flags that are mutually exclusive. For example, the researchers can choose to have a moderator account for which the \textit{isMod} flag is set to \textit{true}. We can add as many of these permission flags as we want to. For the influence analysis experiment, we added the \textit{isPuppet} flag for predetermined sock puppet accounts and the \textit{isObserver} flag for observer accounts that can only read Posts. The limitations of these roles are explained later in section 3.

By utilizing the \textit{disabled} Boolean flag, we can disable or “shadowban” certain Users so they can no longer post, comment, or leave likes.

To create Users, the moderators (researchers) can choose to turn on the sign-up form where the subjects can simply sign up using an email and their chosen username/password combination. On the other hand, the back-end provides an API endpoint through which the moderators can sign up a bulk of users with predetermined roles, emails, usernames, and passwords from a comma-separated values file.
2.1.3.2 FollowerFollowing

I decided to create a separate model to represent a follow (or friend) relationship. This way, I can get timestamps when a relationship started or stopped, which could generally be valuable for analysis.

The FollowerFollowing model simply establishes a relationship between a User who wants to follow or befriend another User. Therefore, the foreign key fields are named FollowerId, which is the id of the User following another User, whose id is kept in the FollowingId field.

2.1.3.3 ObserverObserving

Similar to FollowerFollowing, this model serves to record a relationship when an Observer decided to start following (observing) another regular User.

2.1.3.4 Post

The Post model was modeled after Twitter’s, meaning that the field (property) names of DartPost Post objects match those from Twitter, which are publicly accessible via their API.

The actual content of every Post object is kept in a large string called text. However, the text string holds Markdown to support both microblogging and regular blogging. This way, users can simply format their posts using Markdown and also include images (including animations) by providing the direct image URL. The users can also upload pictures from their computers. A simple script uploads them to our Amazon S3 [1] bucket and then returns the Amazon Web Services link and saves it in the picture_url field.

The User model has a 1 : n relationship with the Post model, meaning that each User can be an author of multiple posts. AuthorId is the name of the field that contains the foreign key (id) of the User who wrote the Post.

The Post model has a 1 : n relationship with the Like, Comment, PostView, and ReportPost models. Moreover, Posts have a 1 : n relationship with the Post model as a Post can have multiple reposts (also known as retweets on Twitter).

Posts also have the field called tags. When creating a Post, you can send a list of words that will be cleaned and kept in an array as hashtags. Due to optimization reasons, I decided not to create a separate model and relation for each Tag. However, these tags
are searchable, and the search bar provided will find each Post with the queried hashtag in the \textit{tags} list.

The \textit{is\_marked} field serves as a Boolean soft-delete flag. If a moderator “marks” a Post, the back-end flips this tag to \textit{true}, and the marked Post will no longer be included in any query.

\subsection{2.1.3.5 Reposts}

Reposts are what Twitter calls retweets or replies. I decided to use the same Post model for Reposts, but I created two foreign key fields called \textit{parent\_id} and \textit{root\_id} to maintain proper relations in a repost chain.

For instance, let Post X be the original post, let Post Y be a repost (also known as reply or retweet on Twitter) of A, and let Post Z be a repost of Y. The chain would then look like this:
Post X would have a repost count of 2, as it is set to be the root_id twice. However, Post Z has its parent_id set to Post Y’s id. You can notice that the front-end is always instructed to embed the original (root) post (Post X in this case) in the repost. However, this can easily be changed in the back-end controller.

### 2.1.3.6 Like

Each Post can have many Likes, but one User can only Like the same Post once. If a User likes a Post, the back-end creates a Like object with the Boolean like set to true and uses the UserId and the PostId fields to connect the Like to the User who created it and the Post it is associated with. If the same User unlike the Post, the related Like
The Like model can be modified to support upvotes and downvotes instead, which is a common feature on platforms like Reddit.

### 2.1.3.7 Comment

Although DartPost supports Twitter-like retweets in the form of reposts, it also supports classical article-like comments. A User can write many comments in general, but also many comments under one Post. These relations are set up using the `CommenterId` (User’s id) and the `PostId` foreign key fields. A Comment object can contain Markdown in the `text` field, so these comments can also be formatted and can include images from URLs.

### 2.1.3.8 ReportUser and ReportPost

These two models can be useful in many OSI experiments as they can be precisely customized, which is not a common feature for Facebook or Twitter reports. The two models are used to keep records of User reports against another User or their post. They contain an optional textual field called `report`, which can take a comment from the User (Reporter) reporting another User (Suspect) or their Post, and these comments might be useful in NLP analysis.

### 2.1.3.9 SearchQueries

I created the SearchQuery model to keep track of every search typed into the provided search bar in the navigation bar. These SearchQueries also have a timestamp. This temporal data can be useful to see if a User decided to search for something related to a Post we just recorded them reading. A User can have many SearchQueries; hence this is a 1 : n relation established through the `UserId` foreign key field.

### 2.1.3.10 PostViews

The PostView model serves as a record of a User viewing a Post. Every time a PostView is created by the relevant controller, the `id` of the Post that was just viewed is recorded in the `PostId` field, and the id of the User who viewed it in the `UserId` field.
2.1 Back-end

There are five important Boolean fields in this model. Only one is set to true depending on which page was the Post viewed on:

- timelineView - true if the Post was viewed on the User’s timeline
- singleView - true if the Post was viewed on its page (clicked on)
- searchView - true if the Post was viewed as a part of the search results
- userPageView - true if the Post was viewed on the User (Author’s) page
- exploreView - true if the Post was viewed on the trending page

2.1.3.11 Messages and GlobalMessages

These two models represent the Messages that the Moderators can send to Users, either individually (specified in the UserId field) as a Message or to everyone as a GlobalMessage. Both of these Messages are then pinned on top of the homepage. In Figure 2.1, a GlobalMessage is displayed as a blue post-it, and a regular Message would typically be there above it as a yellow post-it.

2.1.4 Controllers

2.1.4.1 Users

For the User model, the controller method returns sometimes include Posts and sometimes just the User objects. This depends on where in the front-end the method was called.

The disable method provides a soft delete for the Moderators. It can be customized to either disable the users from logging in or just posting.

A big part of the User controller is authentication. DartPost does not store plain text passwords in its database as an important security measure, only their encrypted hashes.

The back-end performs authentication using JSON Web Tokens (JWT) [2] instead of server-side sessions. First, I create a secret environmental variable that is used to sign JWT’s on the server-side. When a User is signed up using the signup method, a Sequelize pre-save hook generates salt using the bcryptjs package [11], hashes the password, and saves it into the database. When a User tries to sign in, the signin method generates the salt again using bcryptjs, adds it to the submitted password, and passes it to the
validPassword method. In that method, the hash is then compared with the one in the
database using bcryptjs.compare, and it returns a Boolean true if the passwords are the
same. If the submitted and then hashed password matches the hash in the database, the
signin method will call then encode a new JWT token.

Now, the User’s API tool or the browser can save that token and use it in the
authorization HTTP header. From this point, it is sufficient just to verify the JWT token
using the Passport.js package [34] as a middleware. These tokens can be customized to
expire within a specific period which might be useful in some activity timed experiments.

Following common web development guidelines, I created a higher-order component
called RequireAuth, which restricts access to unauthenticated Users. This component is
then always passed to all the routes that should require authentication.

2.1.4.2 Posts

There are many getter and setter Post controller methods that do not require a detailed
explanation. I can generalize a Post Procedure for those controller methods that return
a single Post or a list of Posts as follows:

1. Check the user role of the User accessing the API endpoint through the request
   object. If they do not have permission to access a certain method, raise an error.

2. Find the Posts (array of objects) that fit the given filter (if specified).

   (a) Perform SQL joins to find the Author, Like, and Comment objects associated,
       and embed them inside the Post object

   (b) Return everything as a JSON array of objects.

3. Asynchronously perform these additional queries for each Post in the array:

   (a) Find the Like and Comment counts for the Post and create virtual fields for
       them.

   (b) Find the parent Post if there is one through the parent_id field and embed the
       parent post object into the children’s Post object.

   (c) Find the number of Reposts of each of the Posts in our array. This is done by
       running a COUNT SQL query to count the number of rows that either have
       the parent_id or the root_id set to this Post’s id.
(d) Mark on which of the five possible pages was this Post requested, and mark it as viewed by creating a PostView (impression) record with the corresponding page Boolean field set to true.

4. Return the final JSON array

This procedure is used very often in the code, and it could be optimized by more precisely caching the Like and Repost virtual fields for counts. When using a controller of a model associated with the Post model, such as the Like, Comment, and User models, that controller will trigger this procedure every time the expected return is a modified Post object. A critical step is 3(c), as this is where we can precisely capture impression data.

Although this seems unnecessary, and a possible optimization would simply be not to do this precise data join across multiple models every time, with a lot of user activity, I saw that the numbers regarding engagement metrics could become very inaccurate within minutes. This is a problem if, for instance, we wanted to see whether like, comment, and reposts metrics affect the way the Users feel about certain Posts. Perhaps the researchers want to test a hypothesis that Posts with many likes might be more influential, so I find data integrity very important at this step.

Sequelize joins are performed with simple include statements that include the names of the associated Models we want to include. The documentation for Sequelize v5 suggests that setting a variable separate to true within this statement might significantly improve performance, and this is something I am currently testing. When using separate, Sequelize will split the JOIN queries into separate SELECT statements and then join the data in code, instead of having the database do it.

One of the most specific Post controller methods is called genExplore. This genExplore controller method selectively selects the top 30 posts determined by the output of an engagement metric formula:

\[ \text{sortCriteria}(Post) = \text{num. of likes} + \text{num. of reposts} \times 0.75 + \text{num. of comments} \times 0.5 \]

The posts can also be filtered to only include those created within a certain time (e.g., last 12 hours). genExplore sorts them descendingly by the sortCriteria and sets their Post model flag is_trending to true. This procedure automatically runs every 6
hours as a UNIX server cron job.

2.1.4.3 Like

The Like model only needs two controllers, the likePost and unlikePost methods, which eponymously either add or remove a Like from the Post’s Likes array and trigger step 3 of the Post Procedure. There is also an option only to return status codes whether a Like was added or removed.

2.1.4.4 Comment

Similar to the Like controllers, the commentPost controller creates and associates a comment with a post. It returns the modified Post object with the associated Comment in the Comments array of the Post object. The deleteComment method destroys (hard deletes) the comment if a Moderator requested that. Otherwise, it returns an error. Notice how there is no method to edit the comment, as I currently do not allow the Users to do that. For a certain experiment, however, I can enable edit methods on all content-related models.

2.1.4.5 Reports

The customizability of the Reports’ controllers is important for the OSI operation experiments that DartPost can run.

There are two eponymous methods, called reportUser and reportPost. If the User was reported by distinct users 3 or more times, they are flagged as suspicious, and the controller updates that field in their User database object. Moreover, this method only allows active regular Users to report (not puppets or observers) other Users. On the other hand, everyone can report posts containing offensive speech or exposing private information.

A future update of this controller can also check what the report comment field contains and send a notification to the researchers or Moderators if something is high-alert.

I implemented the getter methods for both kinds of reports, and these methods are triggered by the Moderators when they use their front-end report dashboard.
2.1.4.6 Search

The search controllers are simple SELECT queries. At the scale we tested the platform (500+ concurrent users), there were no performance benefits of a full-text engine, as Postgres automatically creates indexes.

The `searchAll` method is a Promise JavaScript object that merges the returns of the two methods called `searchUsersByName` and `searchPostsByTag`.

The `searchUsersByName` method returns a list of User objects that contain the queried letters in their username, while `searchPostsByTag` does the same for Posts that contain the mentioned hashtag.

2.1.4.7 Messages

For individual messages to Users, I defined three controller methods, each of which requires Moderator privileges. The `messageUser` sends a message to a specific User, while `sendGlobalMsg` sends a message to everyone. The two getter methods are `getLatestMsg` and `getGlobalMsg`, which return only the latest of both kinds of messages.

2.1.5 Migrations

The `migrations` folder contains the migrations of every model and schema modification. If the Moderators want to customize the models even more or change something during a live experiment, the `sequelize-auto-migrations` package [15] will help generate the migrations file that can easily be executed using the CLI.

2.1.6 Data Export and Integrity

The researchers (Moderators) have direct access to the production database using pgAdmin [32]. It quickly connects to the Heroku Postgres datastore and can provide valuable statistics (Figure 2.4) on database connections and transactions. It also provides a tool that the researchers can use to type in SQL commands and export the query results into a comma-separated value file. This file can then quickly be used with machine learning libraries and pandas [33]. The Heroku datastore automatically takes a backup at least every 24 hours, providing us with continuous protection. This also allows us to quickly rollback the state of the platform when needed.
2.1 Back-end

Figure 2.4 pgAdmin DartPost statistics page

2.1.7 Data Import and Seeding

The seeders folder contains a python script, and CSV and JSON files. Since DartPost User and Post models were inspired by Twitter’s, the JSON objects in this folder can be direct returns of the Twitter API. This means that the script can then import those JSON files, connect to the DartPost back-end API, and quickly import them into our database with no modification.

This allows us to quickly seed a DartPost environment with either real Twitter posts and accounts, or create random accounts and groups from CSV files.
2.2 Front-end

Like the back-end, the front-end app is maintained and version-controlled on GitHub in a repository called `dartmouth-socialplatform` [26]. To write the code, I used ECMAScript 6 and Babel [3] to transpile it to ES5 syntax that most browsers understand. However, DartPost’s front-end was developed to support the desktop version of the Google Chrome browser primarily.

Since my goal was to build a scalable, fast, and static front-end application, I started by writing some boilerplate Webpack code. Webpack [48] is a build tool that bundles all the JavaScript files I was about to write into one minified file, analyzes the code to include only what is necessary, and minifies the HTML and SASS/CSS files for optimization.

The core of the front-end app was built using React.js [39] v16 class-based Components and Redux [42] that manages their states. For styling, I mostly relied on CSS flexboxes and UI toolkits provided by React-Bootstrap [41] and Shards-React [12]. Both of those toolkits are based on Bootstrap [8], Twitter’s official open-source styling framework that is used on a variety of social networks. This way, it will take less time for DartPost’s Users to become familiar with the user interface.

Due to these optimizations, this static application takes only \( \sim 30\text{MB} \) of storage space after compiling and can be hosted completely for free on surge.sh [44] servers pointing to our custom domain dartpost.org.

2.2.1 User Profiles

Every User has a dedicated profile page like the one in the Figure 2.5 that shows their Posts reverse-chronologically. It can be customized to show their email, profile picture, and more. The follower and following metrics are currently clickable and open a list of related Users. Some researchers might decide to hide those lists in certain OSI experiments.
2.2 Front-end

The figure 2.6 shows what a suspicious User profile from our last OSI experiment looks like. Three other distinct Users reported this User for being a sock puppet account, and to regular Users (not to other suspected sock puppets on the platform), the reported User’s Posts could show up with red “SUSPICIOUS USER” labels if that is what is desired in a specific experiment. Researchers can also customize this labeling behavior in other ways for certain experiments.
2.2 Front-end

2.2.2 Posts

Posts can be created by clicking the New Post button in the navigation bar. The link opens a simple form (Figure 2.7) where users can type in their Markdown content and upload an image with a button click. It also contains an input field for tags. The words they type into this field will be shown as clickable hashtags on the bottom of a published Post.
The figure 2.8 shows what a published Post looks like. It renders the Markdown content inside the body of a white paper card. The card’s header contains the username of the Author of the Post on the left side, and if the Post is a Repost, it can contain a hyperlink to the parent Post. The \textit{createdAt} timestamp of the Post is on the right side of the header. The footer of the card contains the clickable hashtags that the Post was tagged with, the engagement buttons (Like, Comment, Repost, Reports, soft delete for Moderators), and their respective counters. The visibility and clickability of each of those elements are easily customizable in React.

The comments are rendered in black, slightly transparent cards, and the Moderators can delete them with a simple button on the right. For some experiments, it might be more suitable to use Reposts as the only form of reply and hide the Comments, while on the other hand, Comments might be more suitable when creating a forum-like discussion. DartPost also supports the use of both simultaneously. Suppose a User clicks on the Show Replies button, all the Reposts (or the latest 10) pop out of a dropdown like in Figure 2.9. Comments are visible by default and are sorted chronologically.
2.2 Front-end

The Timeline, or the DartPost homepage, was mentioned earlier in this thesis and is pictured in Figure 2.1. It is also accessible by clicking on the DartPost logo in the navigation bar. This page can be customized to support infinite loading (scrolling), which would paginate the Sequelize SELECT query to serve about 15 Posts every time the User scrolls past the last visible Post.

The Explore page link sits right next to the logo in the navigation bar, and it takes the Users to a page (Figure 2.10) that displays the 30 most trending posts.

**Figure 2.9 Individual Post Reposts/Replies**
2.2 Front-end

Figure 2.10 Explore (Trending) page (http://dartpost.org/posts/explore)

2.2.3 Reports

The exclamation button in the Post card footer opens the report modal (Figure 2.11) to report a Post. The modal can be customized with different reporting options, and the one that a User chooses can be saved in the custom report field of the new ReportPost object. The modal can also include an input box (Figure 2.12) which the Users can use to type in a specific reason why they are reporting. This is useful in some OSI experiments, as the researchers might prefer comments instead of offering generic options for reporting sock puppets accounts.
The moderators have access to two tables that display the ReportPost and ReportUser data, both accessible from the Moderation dropdown in the navigation bar. For instance, the Figure 2.13 shows the ReportUser table that links the Reports with the involved Users.
2.2 Front-end

![User reports]

**Figure 2.13** Table of reported Users available to Moderators

### 2.2.4 Search

Users can use the input box in the navigation bar to either query for usernames like Jack or posts with hashtags like #covid (with or without the # sign). The input is then cleared and passed to the Search controllers that combine the results. The figure 2.14 shows the results that come out when a User searches the current database for “Jacob.” I display a card with some simple User information and the Follow/Unfollow button for each User result.
When a User searches for “#covid” or “covid,” there are no User results, so the Search page only displays Posts that include that hashtag (Figure 2.15):

**Figure 2.14** Search results - Users

**Figure 2.15** Search results - Posts
2.3 Metadata

DartPost is connected to four additional apps that provide behavioral analytics (HotJar, Google Analytics [21, 17]) and server logging (New Relic [18], Papertrail). These applications can be turned off, depending on the IRB requirements and the information provided in the consent forms.

HotJar data could help the researchers understand how visitors (Users) use DartPost by providing session recordings, i.e., screen recordings of their typical sessions. These recordings are helpful when trying to improve user experience, but they also tell us how much time Users spend on particular pages and provide us with the context of how they move away from them. HotJar also provides three types of heatmaps. One of them tells us where Users most commonly click, one tells us how their mouse moves over the page, and the most important one tells us how far they scroll on certain pages.

For instance, if the scrolling heatmap (figure 2.16) told us that 50% of visitors don’t make it past the fifth post on their timeline, the researchers could use this data to penalize posts that show up too often or promote posts that do not show up enough.
Figure 2.16 An example of a HotJar scrolling heatmap for the DartPost Explore page. The area that most Users saw in their viewport is marked as hot. Typically, the hot area is much larger than in this sample figure that just displays the heatmap’s colorway (yellow - hot, blue - cold).
2.3 Metadata

Google Analytics provides us with demographic, geographic, and technological data about our Users. Commonly, experiments on DartPost will recruit subjects from Amazon MTurk and similar sites, but if those sites do not provide such information about the subjects, Google Analytics makes it easy to acquire. The researchers’ consent form should highlight that DartPost can collect this, as in some cases, IRBs classify such data as personally identifiable information (PII).

Although I use the React Router `history` package [40] to manage the Users’ browser session history, DartPost does not store that data. Instead, I utilize Google Analytics to monitor the Users’ behavioral flow. The graph in the figure 2.17 tells us how they navigate through pages and travel from one Post to another. This data reinforces the impression data collected through PostViews, and helps the researchers understand what kind of content makes the Users navigate to other similar Posts.

![Google Analytics Behavioral Flow graph](image)

**Figure 2.17** Google Analytics Behavioral Flow graph

Google Analytics data is also useful when combined with the data from New Relic, as together they can tell us the page loading times in different parts of the world, and which pages are typically most requested by the Users. Then, we can optimize the platform for a better user experience, which could diminish at a large scale if not dealt with properly.

Furthermore, researchers can use such data to determine the adequate compensation for the subjects based on their activity on DartPost. Google Analytics can tell us their average daily session duration, page views, and the number of sessions per day, all-
important activity metrics (Figure 2.18).

**Figure 2.18** Google Analytics dashboard for DartPost
Chapter 3

Use Case

Using Impression Data to Improve Models of Online Social Influence

The biggest goal of DartPost is to provide a platform that enables others to create different types of experiments to study social influence operations that are impossible to run on real work social networks such as Facebook and Twitter for one or more of the following reasons: (i) they violate the norms and rules imposed by the platform, or (ii) they do not acquire IRB-authorized informed consent from subjects, or (iii) they require interventions from the platform. DartPost seeks to overcome these roadblocks. In this section, I describe one such experiment that was enabled by the DartPost platform. I was not the principal author of this experiment, and I do not claim its results as part of my thesis, so this section explains how DartPost can be used in a real-world simulation [24].

In September 2020, we designed and ran a social media simulation (with IRB approval) to test DartPost. DartPost enabled us to run this experiment as the first of what may be many such experiments. The goals of this work were to study influence, the ability to change the beliefs and behaviors of others on social media. Past studies were limited by publicly available data that record expressions (active engagement of users with content, such as likes and comments) and neglect impressions (exposure to content, such as views). Moreover, they lack ”ground truth” measures of influence, such as information about sock puppet accounts that were explicitly created for deception. The paper demonstrates that
impressions are much more important drivers of influence than expressions.

My contribution to the paper was developing DartPost and tailoring its features to the requirements of the experiment, particularly regarding impression data collection by creating the PostView model. I helped seed the database with accounts that were to be used by the subjects, and ensured that the platform was running smoothly throughout the five-day experiment. I converted the collected SQL data into CSV form and provided it to the rest of the team for analysis, summarized below.

For this simulation, we decided to create four different account types, and to create them quickly, we used a script that passed comma-separated values to the signup API endpoint. The different account types we created are:

**Observers**

- Observers can follow other accounts. They cannot post, like, or reply
- Observers cannot report accounts as sock puppets, but they can report speech violations.
- When other users click on observer accounts, they cannot see any information about them
- \( isObserver \) set to \( true \)

**Active Users**

- Normal user accounts such as those on Facebook and Twitter
- They can post, like, reply, and report other users for hate speech or being a sock puppet

**Sock Puppet Masters**

- Assigned several user accounts
- All capabilities of Active Users, but cannot report other users for being sock puppets
- They can report speech violations
- \( isPuppet \) set to \( true \)
Moderators

- Accounts for the researchers
- All capabilities as Active Users, but can quickly access data on Post and User Reports
- They can soft delete posts and users
- $isMod$ set to true

The simulation recruited a set of 287 users from Amazon’s Mechanical Turk crowdsourcing service to use the platform over the course of five days. Only American citizens over the age of 18 were included. Users who consented to participate in the study provided us with their Mechanical Turk worker number, which is not linked publicly to names or other identifying information. Participants were paid 10 USD per hour and were expected to use the platform between 15 minutes and 1 hour per day, depending on their role in the simulation.

Each user was assigned to one of three main roles: 160 were assigned to be single account operators (active users), 40 were assigned to be multiple account operators (sock puppet masters), and 87 were assigned to be observers. Multiple account operators were then randomly assigned a number of unique accounts between 4 and 8. Users did not know the nature or distribution of these roles across other users or that some other users were operating multiple accounts. For each role, instructions were given in the form of a unique document linked in their DartPost navigation bar.

Single and multiple account users were each assigned positions on three of the following eight contemporary political questions: (1) whether the U.S. government had done too much or not enough to combat COVID-19; (2) whether the government should have more authority to regulate social platforms such as Twitter and Facebook; (3) whether the environment or the economy should be given priority in environmental policy; (4) whether they approved or disapproved of the “Medicare for all who want it” health care system; (5) whether they approved or disapproved of a two percent wealth tax on people with more than 50 million dollars in assets; (6) whether they believed that foods containing genetically modified ingredients are safe and healthy to eat; (7) whether they agreed that the United States should pay less attention to problems overseas and
concentrate on problems at home; and (8) whether they favored or opposed an increase in the number of nuclear power plants in the United States to provide electricity.

Users were instructed that “your task is to get other users to view and like your posts and to convert them to your positions on the issues. You may do this by writing posts or posting links to relevant content on the Web, reposting other users’ content on DartPost, or replying to or liking other posts.” To incentivize users to actively seek to influence other participants, we informed them that they would be given a bonus payment that would increase “the more users who view, like, and repost your posts, and the more users you convince to support your issue positions.”

Observers were assigned passive accounts. These accounts had the capability to follow other users but could not like, post, repost, or comment. Unlike the other users, observers were not assigned positions on any of the eight issues. Observers’ instructions stated “Your job is simply to observe what other users are doing on the platform. These users are trying to maximize positive exposure and agreement with the positions they support on several key political and social issues...”

Before each user was assigned a role at the beginning of the simulation, and then once every 24 hours after completing their minimum time using the platform, all users were asked to complete a survey. The initial survey collected standard demographic data, including age, race, gender, educational attainment, political affiliation, and information about the subject’s use of social media. All subjects were then asked to indicate their position on the eight political questions described above. On each subsequent day, users were asked the same eight questions again, allowing us to track changes in opinions over the course of the simulation.

Although users knew they were participating in a simulation, most agreed that the experience faithfully represented social media use in the real world. At the end of the simulation, 86 percent of subjects reported that DartPost was somewhat or highly realistic.

The paper makes two major contributions that were not made by me and are not the focal point of this thesis. First, it defines the novel concepts of Direct Influence Networks (DINs) and Full Influence Networks (FINs) that trace back the impression or expression events that precede a change in opinion by an observer. We propose three influence models, FI1, FI2, FI3, that leverage both expression and impression data and build on
FINs to create a complete picture of the mechanics of social influence.

![Diagram of a Sample Direct Influence Network (DIN)](image)

**Figure 3.1** Figure 16: Sample Direct Influence Network (DIN) for subject $S$ with polarity $P$. This sample DIN is associated with an observer $A$ (purple node) at the beginning of day 5. On day 4, there was an impression event (node $e_1$) in which that observer was served a post $P_1$ by account $B$. Thus, observer $A$ may have been influenced by post $P_1$. Three events are reported on day 3 and raise interesting questions. Should account $B$ (that posted post $P_1$ on day 3, see event $e_2$) receive credit for influencing $A$? But on day 3, $A$ also viewed posts $P_2$ and $P_3$ that contained the same subject and polarity as $P_3$ (events $e_3$ and $e_4$ respectively), which were posted by account $C$ and $D$ respectively on days 1 and 2.

The second set of contributions involve 5 novel findings, strongly tied to DartPost’s ability to collect impression data.

(i) We demonstrate that impressions, available in the PostView table, account for far more exposure to information on social media than do expressions and that impressions and expressions are not highly correlated. Yet, for the reasons described in the introduction of this thesis, impressions have been largely ignored by the majority of the literature to date.

(ii) We find that once participants are influenced to adopt a position on a particular subject, the diversity of positions on the same subject they choose to view (as measured by entropy) decreases dramatically over time. This suggests the potential for influence operations to polarize social media users through the formation of echo chambers. Though the existence of echo chambers has been described before [5, 16], to the best of our knowledge, we are the first to document the emergence of echo chambers using both impression and expression data from a social platform.
(iii) Our proposed models of influence accurately capture acts of successful influence in the ground truth generated by our simulation.

(iv) As expected, only a small minority of accounts were identified by observers as being influential. However, we found that accounts that users did not consider influential at the beginning of our study remained largely un-influential throughout the study. In contrast, users who were influential at the end of the study were likely to have been identified as influential from early on.

(v) Using quasi-Poisson and Gamma-Poisson regression models to identify the determinants of influence, we found that all expression types (posts, comments, likes) and impressions are individually statistically significant and positively linked to influence at the $p < 0.01$ level. When we consider all these features simultaneously, however, only the impressions are positively linked and significant. Specifically, our regressions show that when the impressions of a particular account on the platform increase by one standard deviation, the expected number of times users cite the account as influential more than doubles. These results suggest that the single biggest determinant of influence on social media platforms is impressions, a factor that has hardly been studied in the growing OSI literature.

There are several important limitations to our study. First, the “in-vitro” nature of the simulation can have an effect on the behavior of the participants. Though our users confirmed that our simulation faithfully replicated the environment of prominent social media platforms, our participants’ motivations were primarily monetary, whereas social media users in the real world are more likely to be a mixture of social and political. Likewise, because our participants did not reflect a representative sample of social media users, it is possible that their behavior might differ in important ways from users in the real world, particularly very high-profile users who we might expect to have the greatest influence. Second, our simulation lasted only five days. This was long enough to produce valuable data but not long enough to capture dynamics of influence that may take longer to develop. Third, the “natural” environment in a social media platform is saturated with millions of posts discussing a large variety of topics, while in our simulation, the platform was not pre-populated with any posts.

Given the importance of the impression data and measurement of ground truth our methods make possible, we believe the limitations of our approach are far outweighed by
its advantages. Because social media platforms do not make impressions data publicly available and because researchers cannot control many critical aspects of the social media platforms they study, simulations like the DartPost-enabled methods in this study provide a valuable tool for researchers. Future work can address some of the limitations of the present study by, for instance, conducting longer simulations or by pre-populating DartPost with real-world content that is publicly available from social media platforms, such as Twitter. Once more, the contributions of the experiment mentioned in this section were not mine alone and are not the centerpiece of this thesis. Still, these kinds of findings would have been impossible without the use of DartPost that I solely implemented.
Chapter 4
Conclusions

Our influence experiment in September proved that DartPost could successfully be used for influence operations studies. We can control many critical aspects of the platform and run a variety of different DartPost environments at the same time, making it a valuable tool for researchers. With minor modifications, DartPost environments can also be pre-populated with real Twitter posts, providing a significant kick-start to many social network experiments. Besides the expression data, DartPost also provides critical impression data via the PostView table.

Implementing infinite scrolling on all components that display an array of Posts would help improve the data quality. When infinite scrolling is on, it increases the researchers’ confidence that the User saw every Post that they scrolled over. To further improve the quality of the impression data, I could implement a timer in React that times how long a certain Post (a React component) has been in the User’s viewport. We could define this time as a “Post reading time” and create a new database model that keeps track of each reading duration and time of day and connects those to the Post and the User who read it. Some OSI operation experiments might benefit from how much time a User spent reading a certain influential Post and comparing it to the average time that all Users spent reading that Post.

DartPost also opens up a wide range of avenues for further research. For example, researchers can use the platform to study how humans and social media bots interact, and how different types of bots or levels of bot activity shape network formation and user opinions. Researchers can also explore how disinformation on social media affects users, identifying the conditions that make it more likely to succeed or fail, and whether high levels of misinformation might cause users to distrust even factual information they
encounter on the platform.
References

[33] pandas - Python Data Analysis Library. URL: https://pandas.pydata.org (visited on 05/17/2021).
[34] Passport.js. URL: http://www.passportjs.org/.
REFERENCES


[37] PostgreSQL: The world’s most advanced open source database. URL: https://www.postgresql.org (visited on 05/17/2021).


[40] React Router: Declarative Routing for React. history. URL: https://reactrouter.com/web/api/history (visited on 05/20/2021).


