UnFold: A Modern Payment Platform for Publishers

Sweyn Venderbush
Department of Computer Science
Yale University
Advised by: Ruzica Piskac
sweyn.venderbush@yale.edu

Abstract

The internet has dramatically changed the way that people discover and consume news media. The average consumer reads a larger number of news articles from a wider variety of sources than ever before. But models of monetization haven’t shifted to reflect this new paradigm of consumption, instead relying on the 20th century models of monthly subscriptions and advertisements. These models have proven to be ineffective in balancing content accessibility with the need for publishers to finance high quality journalism. This project aims to offer a proof of concept for a new model of monetization for web publishing that is consistent with modern consumption patterns. It allows publishers to price their content on a per article basis and gives users the ability to pay for articles by any in-network news source with minimal friction and no transaction cost via a centralized account, which also keeps track of every transaction and all the articles a user owns so they can easily come back and read them again in the future. As a result, for the first time, users can quickly and easily financially reward publishers for content they find valuable, wherever it is on the web. The deliverable technical work consists of three pieces: a backend component that keeps track of user information, publisher accounts as well as associated articles, and transactions on the network, a front-end component that allows publishers and users to manage their accounts and allows users to add money to their account and transact on the network, and a backend plugin for Django-based web applications as a proof of concept of how UnFold could be easily integrated into a publisher’s existing site.

1 Introduction and Motivation

UnFold is a web platform that allows users to pay for web content on a per-article basis. Any publisher can quickly and easily integrate UnFold into their existing site, set prices for their articles, and allow users to transaction safely and securely using their single UnFold accounts, minimizing transaction friction. This model offers a viable alternative to the existing models of monetizing web publishing content, advertising and paywalls, which are antiquated and becoming increasingly ineffective, resulting in publishers having difficulty financially sustaining the work needed to result in quality journalism.
Since the first banner advertisement ran on Wired.com in 1994, advertising has been billed as the monetization strategy of the future for publishers [0]. In recent years, however, the landscape of advertising has changed dramatically. Publishers have come to the realization that advertising revenue is not sufficient to sustain publishers, especially top newspapers, which have larger news teams and spend more on daily global coverage and investigative journalism. This issue has been further exacerbated by the emergence of ad-blocking software in recent years. 40% of US desktop users and 15% of mobile users used an ad-blocker in 2017 and that number is continuing to rise dramatically [1]. That number is even higher in Asia. There are also a number of hidden costs of advertising as a revenue model that publishers are beginning to understand the downsides of. In particular, ads are known to add significant latency to the webpage [2]. Especially in an era where users are browsing quickly between many different articles, long load times can be enough to cause the user to bounce from the page and find a different article. This is especially true on mobile, where bandwidth and rendering capacity are especially limited. A final issue with ad paradigm is its reliance on personal information to target ads. There has been significant pushback on the way that ad giants like Google and Facebook collect personal information about users to serve them the most effective advertising. On the other hand, if an ad isn’t well targeted, it is less valuable to the advertiser and as a result, publishers are willing to pay less for it, resulting in a catch-22.

As a result of these issues with advertising as a revenue source, many publishers, especially legacy publishers who started by publishing physical paper newspapers and magazines, have moved to a paywall model. Over 75% of top US newspapers had implemented some form of paywall by 2015 [3]. While these legacy publishers have been relatively quick to adopt paywalls, digital-only publication have been much more reluctant to do so, realizing their core reader bases are less likely to be willing to subscribe given both the demographics at play and the types of content that these sites tend to produce versus larger publications like the *New York Times*.

This monthly subscription model is fundamentally antithetical to the way that readers behave and consume content on the internet and is instead a direct translation of the 20th century model of paper subscriptions. Prior to the internet, people had to commit to receiving a physical paper every morning as their main news source. As a result, they might subscribe to one or maybe two papers a month. On the other hand, the internet has resulted in what has been referred to as the "horizontalization of media" [4]. People read many more articles from a much wider varieties of news sources and are no longer locked into one or two major papers. The advent of search and social media has made discovery of smaller sources of content significantly easier. People may read multiple stories on a single topic in order to gain a variety of perspectives. Top investigative journalism pieces come from the *New York Times* and *Wall Street Journal* but also from *The Atlantic* and *Buzzfeed*. 
Millennials are especially likely to distribute their consumption habits in this way and as a result are even less likely than average to subscribe monthly to a news publication or app [5]. This should be especially troubling to publishers since, given demographic shifts, these will soon be the main consumers of web content in every vertical.

While monthly subscriptions are a vestige of a pre-web era, microtransactions are fundamentally a part of the internet. iTunes taught us to pay per-song rather than by the album. We now buy $.99 games for our phones more than we do $60 games for our gaming consoles. UnFold looks to bring this same paradigm to web publishing, allowing anyone to be rewarded in a straightforward way for the content that they create online.

2 Functionality Overview

The UnFold prototype developed for this project consists of three parts:

- A backend application that manages user accounts, balances, credit card transactions, and article purchases, as well as makes this information accessible to relevant parties via a set of internal and RESTful APIs and manages permissions for accessing this information.
- A frontend application that allows users and publishers to create and manage their accounts, including balances and previous purchases, add money to a user’s account to be used to purchase, and access any articles a user has previously bought.
- A backend plugin for Django-based websites that serves as a means for any publisher to easily integrate the UnFold payment interface into their existing site by implementing one short function and decorating the relevant Django views that generate the articles on the site. This library aims to serve as a proof of concept of how easily UnFold might be integrated into a web server with any backend framework. To fully productize the UnFold concept, it would be ideal to build out similar backend plugins for other popular backend languages and frameworks, including PHP, Node.JS, and Ruby on Rails. Even before these additional frameworks are built, any publisher could write their own UnFold integration using the existing authentication flows and accessible API endpoints, as I have done to create the Django plugin. However, these flows and endpoints, while complete and secure, are not currently formally documented.

The work for this project is delivered via two codebases: the **unfold** repository contains the backend and frontend pieces, which allow users and publishers to manage their UnFold accounts and transact on the network. The **unfold-plugin-django** repository includes the code base and documentation
for the UnFold Django plugin. I have also included two deployed sites where a user can test UnFold in a demo environment. The first site is a deployed version of the UnFold frontend, which allows publishers and users to manage their accounts and transact on the network. I have also provided a simple demo blog to illustrate the ability to transact and to show how easy UnFold is to integrate into any application. These links can be found in the Code Repositories and Associated Links section.

The current UnFold implementation provides all necessary flows for publishers to integrate UnFold into their applications and collect payments, as well as for users to create accounts, add to their balance, and purchase articles from any site that has integrated UnFold.

2.1 Publisher Flows

Publishers can first visit the UnFold homepage, where they can click a link to register for an UnFold Publisher account. During the registration flow, they create a username and password as well as provide an associated email address and the name of their publication to be shown to the user during purchases. After completing the sign-up form, the publisher must confirm the email address they provided by clicking on the link in the confirmation email sent to them. After they do this, they are given the opportunity to login to their account. Upon logging in, they are presented with their profile, which allows them to view and, in some cases, edit relevant details about their account, including email addresses associated, their publication name, and their current account balance.

![UnFold Profile Screen](image)

Figure 1: The Publisher profile screen allows a publisher to easily access and edit elements of their account.

On the profile screen, the publisher is also presented with their UnFold API key, which can easily be copied to their clipboard to be added to their codebase. They can also generate a new API key from this screen if their key is compromised for any reason.
Using this API key, publishers can begin the process of integrating UnFold into their application. This process is incredibly straightforward. Publisher can add the UnFold plugin to their applications by adding the necessary line to their requirements.txt file:

```
-e "git+https://github.com/wesny/unfold-plugin-django#egg=unfold-plugin"
```

The plugin can then be used to easily decorate any views that the publisher wishes to charge for. To do this, first the publisher adds their username and API key as variables in their settings.py file. The publisher then implements a function based on a provided header, which provides UnFold with the necessary information about the article to be displayed. This function as specified is passed all the of the same information as the view, allowing the publisher to identify the article the same way the view does. The function is very flexible, allowing for a wide range of implementation strategies. For example, a publisher might set a static price for all articles, store and query the price for each article as a element of the Article model, or store prices in a separate model from the rest of the Article model and query it separately.

This is all that needs to be done for a publisher to integrate UnFold. They can now redeploy their application and let UnFold handle the process of managing transaction and paying the publisher accordingly.

### 2.2 User Flows

A user can begin the registration process for UnFold in two possible ways: they can visit the UnFold homepage and create an account themselves or they may encounter UnFold by attempting to access an article that requires payment via the UnFold network. Regardless of how the user enters the sign-up flow, the process is the same. The user provides the same information a publisher does (username, name, email, password) but also provides a credit card to be put on file in their account. They also
have the option to seed their account with a beginning balance (if they don’t, their account will have a balance of $0.00 and they’ll have the option to load their account with a balance before making their first purchase).

After confirming their email address, the user can log into their account and is presented with their profile screen. Here they can manage elements of their profile. Most notably, they have the ability to manage payment details, including adding new credit cards to their account and editing or deleting existing ones. They can see a history of all of their credit card transactions and all the articles they have purchased and easily access them from one screen. The user can also reload their account balance. When they do so, they receive an email invoice for the transaction.

After creating an account and loading their balance, users are free to use their UnFold account to purchase articles anywhere on the web that accepts UnFold. To do this, a user can navigate to a site that accepts UnFold and click on the article they wish to purchase. They will then be direct to the UnFold site, which will tell them the article they are purchasing, the name of the publication, and the price of the article as well as their balance. From this screen, they can confirm the transaction and be redirected back to the original site. If the user isn’t logged in to their UnFold account when they’re redirected, they are given the opportunity to do so before being asked to make their purchase. If the user’s account balance is lower than the price of the item they would like to purchase, they are given the opportunity to reload their account before being forwarded on to the purchase screen. Because the entire transaction process happens on the UnFold site, no user information is ever exposed to the publisher’s site besides the username of the purchaser. As a result, users can be sure that their information and account balance are secure. This also makes it impossible for a site to make a transaction on the UnFold network without the user’s permission, which is an important element of a network where publishers aren’t required to become trusted in order to utilize UnFold.

3 Methods, Implementation, and Challenges

3.1 Language and Framework Choices

The first major design consideration I had to make was which backend language and framework to pick for the project. I had familiarity with backend Python web development but had mostly worked with Flask in the past. In order to gain a new skill for this project, I decided to use Django for the backend development. Django also has the advantage over a micro-framework like Flask of having a large number of built-in features that force developers into habits like specific code structuring that make management of large projects much less unwieldy. There is also a large developer community, which provided a number of useful libraries that provided skeleton versions out of the box of some
of the functionality I needed, including user accounts, sign-in flows, email verification and sending, Stripe integration, and single sign-on. I had also worked to some extent with the Django REST Framework before and so thought it would be a good fit for the APIs I would need to build for UnFold. Django also seemed like a good choice for which to build a backend plugin for UnFold as a proof of concept of how to integrate into a publisher’s website and so I had the advantage of leveraging what I had learned from the backend development when I was building the plugin. I elected to use the Cookiecutter Django base template for my project, which provided a basic Bootstrap interface on which to build as well as basic account management tools out of the box that I could augment as needed. It also made it relatively easy to set up a development environment, since it provides the ability to manage development variables and settings separately from production variables as well as a means of deploying to Heroku with relative ease.

3.2 Purchasing Flows

The next major design decision I had to make was around how UnFold would integrate with publisher websites and how transactions would occur. I considered three possible options. The first was to allow
publishers to manage the entire process of transacting from their site so the user would never have to leave. This had the appeal of being the approach with the least built-in friction. However, from a security perspective, it had the downside of giving a large amount of trust to the publishers, which was fundamentally something I didn’t want to do since I had designed UnFold in theory to be a trust-less network (any publisher could sign up and use it to collect payments without having to be verified by a central source). Unfortunately, this model would allow for scenarios like a publisher charging a user without notifying them or charging them a different amount than the amount they were notified of. As a result, despite its appeal, I rejected this approach. The second approach was to require the user to install a browser plugin to use UnFold. This is similar to what many Blockchain-based payment platforms have done. However, this has the drawback of adding friction to the registration process by making the user install such a plugin. The user also couldn’t use UnFold in environments where they couldn’t install a plugin, such as a shared computer or a mobile device. As a result, I opted for the option I ended up implementing, in which the user is redirected through the UnFold site to transact and then returned to the publisher’s site to read the article. This method, although slightly higher friction because of the additional screen that must be showed, means the publisher never has to access any information about the user besides basic identifying information like a username. It is also a method that has been used by top third party payment services like PayPal, which lent it some credibility in my mind.

After making this decision and implementing the basic payment flows, I had to consider a large number of possible edge cases, which make the payment process complicated. For example, a user might click on an article that requires UnFold but not have an account. They might have an account but might not be signed in. They also might not have the necessary balance to pay for an article. All of these required separate user flows but it was important that the user always eventually reached the payment screen and, eventually after confirming the purchase, the article they had originally attempted to purchase. As a result, I had to build the ability for redirect links to be passed from screen to screen as the user navigated all the possible paths to an eventual purchase.

The last design consideration was how to process payments. My original approach was to use a Blockchain-based technology, either an existing currency like Ethereum or building my own token for the project. The advantage of this approach would have been that user accounts could have been decentralized meaning conceptually the network could be truly trust-less and there wouldn’t have to be any backend database managing accounts and payments. The major issue with this approach is that microtransactions on Blockchains are incredibly resource intensive and inefficient, often costing more to transact than the actual transaction. Many people have tried to build microtransaction platforms, including off-chain solutions on top of coins like Ethereum but no one has created a viable solution
to date. As a result, I decided to go the more traditional route of collecting credit card payments via a centralized platform. I integrated Stripe as my means of collecting payments (the deployed demo version of UnFold has Stripe integrated but is set to demo mode so no real transactions will be charged).

The credit card route posed a number of challenges of its own. Just as Blockchains are not good at microtransactions, neither are credit cards. In fact, the minimum charge that Stripe allows is $0.50 and the fees associated with a large that small are high enough that even that isn’t very financially feasible. As a result, I decided to go for a pre-paid model, in which the user fills their account upfront with a larger sum and that balance is then debited from every time a transaction occurs. The other challenge of utilizing credit cards is that the transaction process is complicated and many things can go wrong in the process of transaction. For example, a card may have been canceled since it was added to an account or a change may be declined for any number of reasons (or for no apparent reason). To make this slightly simpler for the user, I required that the user add a valid card during the sign-up process. It is still possible that this card becomes invalid after it has been added. Luckily, Stripe provides a well-documented set of exceptions for the variety of things that might go wrong during a transaction. I had to catch all of these exceptions and generate flows to handle each of them in such a way that it didn’t derail the purchase process.

3.3 Publisher Authentication

The second major area of design considerations I had to make were around authenticating publishers with UnFold and users with publishers. There were two main reasons that this was complicated. First, I designed UnFold to be a trust-less system, meaning publishers didn’t have to be approved in order to use the platform. Second, I wanted to ensure that, despite the trust-less nature of UnFold, there was no way for someone to spoof a user account to a publisher and as a result gain access to articles without paying for them. To do this, I employed a standard known as Single Sign-On (SSO) using the Python library Django Mama CAS.

In SSO, when a user wants to authenticate, instead of creating an account on each individual server or even providing the user credentials to the given server, the user is instead redirected to a third-party service which, presumably, they already have an account with (if not, they can make an account). The user signs in with the centralized service, verifies that they want to provide access to the third-party service they are trying to use, and then are redirected back to the third-party service along with a token. The third-party service can then verify that this token is valid and upon validation, is provided some set of information about the user, potentially a username and name of the user.
I utilized SSO in UnFold as a means of making sure that users couldn’t spoof their identity to publishers but also without having to have the user provide credentials to any publisher. I didn’t want to require a user to go through a full OAuth process for every publisher, which felt unnecessary and would have been higher friction. Instead I implemented the following SSO process. Whenever a user purchases an article, when the user is redirected back to the publisher, the GET request includes a unique, one-time token generated by SSO as a URL parameter. The reason that simply passing back the username of the user was insufficient was that this parameter could be easily spoofed by a user to impersonate another. Instead, once the publisher receives a token, the UnFold plugin library automatically queries UnFold using the token to validate it and is returned the username of the user. From there, that username can be saved in the publisher site’s session and used to authenticate the user in the future. Using this method, the only trusted source for both the user and the publisher is the UnFold server and no one can impersonate anyone else without compromising the central UnFold server.

3.4 Publisher Plugin

I also had to consider the easiest way for a publisher to integrate UnFold into their applications. Luckily, the plugin ecosystem for Django apps is relatively mature, so I was able to take advantage of some templating when it come to putting together my package so that it was pip and easy_install ready. Using a decorator seemed like the obvious choice since it meant that the publisher wouldn’t have to change any of their view code and could easily use UnFold to require payments for many views (for example, a larger site like the New York Times might have one view for standard articles, another for articles with significant multimedia content, etc.). I had never written a Python decorator before so I had to make sure that my implementation could be used across any type of Django view (both class-based and function-based) and that the publisher could easily provide UnFold with the necessary data so UnFold knew what article was being purchased and for how much. There was also the question of the best way to check whether a user had access to an article. In the end, I elected to store the user’s profile locally in the publisher site’s session. If such a profile existed, the plugin would check to see if the user associated with the profile already owns the article. If so, the user is immediately redirected without having to ever leave the site. If there is no stored profile or the user doesn’t own the article, the user is redirected to the UnFold site to complete the transaction. Upon transacting, the user is returned to the original publisher’s site with a token that the publisher can use to authenticate the user (see Section [3.3] for more details). The only known issue with this method is that once a user has logged into a given site, there is no way to log the user out until the user desires to view an article they haven’t previously purchased. Further UI work on the publisher side would be needed to be done to solve this issue, as documented in Section[4]
3.5 Deployment

All code repositories for the project were managed using GitHub during the course of development. I then deployed both the main UnFold server as well as a sample blog with UnFold integrated on free Heroku instances along with the Heroku Postgres plugin with the Hobby Dev plan, which provides a minimal amount of storage. The sample blog I used was Django-Blog-It, which I chose because it was complicated enough to illustrate how easily UnFold could be integrated into even a more complex existing publication. I did not make any changes to the sample blog besides adding the UnFold plugin and annotating the appropriate views along with adding in the single information-providing function. Heroku free instances have limited uptime so if no one has visited the site in a while the instance will go to sleep and may take a few seconds to wake up when visited.

4 Future Work

While the current work on this project act as a valuable proof of concept for the UnFold framework, there are a number of improvements necessary if one were to fully productize and launch UnFold to the public.

The largest area of additional work that would be needed to be done is around integration into publisher websites. Currently, the experience of engaging with UnFold from a publisher’s site isn’t the most intuitive. The current plugin doesn’t provide out of the box a way for the publisher to explain to the user that the site uses UnFold to purchase articles. While the publisher could add the UI for notifying a user separately, it would be nice to provide this out of the box via the plugin. Additional explanation text on the UnFold purchase site would also be beneficial to any users that don’t see or fully understand that the publisher they were using uses UnFold or how it works. One more element of the publisher experience that still needs to be implemented is the ability for publishers to cash out their balances, which should likely occur via bank transfer. Finally, the UnFold plugin currently only exists for Django. A productized version would have plugins for most modern backend languages and frameworks.

Additionally, as mentioned prior, UnFold is a complex application with a large number of possible user flows. Many of these are handled in the current implementation but some of the less likely flows are handled generically with catch-all errors. Before UnFold is brought to market, it would be beneficial to do a sweep of the application, find possible sources of errors, and ensure that, when such an error occurs, the user is redirected to the correct place in the application to resolve it. This is especially true of the current Stripe integration. While the current process is consistent with Stripe’s own best practices and is secure as a result, there are many possible errors as a result of using
credit cards in production, which currently all have the same handler. Separating these out would create a much better user experience around credit cards.

Finally, UnFold was designed with security in mind, both for users and publishers. Still, it would be important to do a full audit of the entire system to find possible sources of insecurity that I haven’t found or accounted for before UnFold becomes responsible for any amount of real money.

5 Conclusion

The UnFold project aims to offer an alternative to antiquated means of monetizing news media on the web. In an era of fake news and misleading headlines, access to quality content is now more important than ever. Providing viable monetization strategies is a key piece of top quality publications being able to continue their high quality reporting. UnFold aim to provide a means of monetization in a way that mirrors modern consumption habits.

6 Code Repositories and Associated Links

- Main Repository for UnFold Server and Frontend: [https://github.com/wesny/unfold](https://github.com/wesny/unfold)
- UnFold Publisher Plugin for Django: [https://github.com/wesny/unfold-plugin-django](https://github.com/wesny/unfold-plugin-django)
- UnFold Frontend and Server Demo: [http://unfoldapp.herokuapp.com](http://unfoldapp.herokuapp.com)

7 Acknowledgements

Thank you to Professor Ruzica Piskac for advising me throughout this project. I appreciate your enthusiasm for the concept that I came up with and your willingness to always push me both conceptually and technically.
References


