

# Wildcard: End user modification of web applications through a data table

Geoffrey Litt<sup>a</sup> and Daniel Jackson<sup>a</sup>

<sup>a</sup> Massachusetts Institute of Technology

**Abstract** Many people use Web applications that do not exactly meet their unique needs. While the Web platform supports client-side modification through user scripts and browser extensions, most people do not have the programming skills to implement such modifications.

In this paper, we present a prototype of a browser extension called Wildcard, that empowers users to casually tweak web applications without programming. Wildcard shows the main data from a web page in a table, and maintains a bidirectional connection between the table and the original page. By directly manipulating the table, people can perform a wide variety of modifications: sorting/filtering content, adding private annotations, using spreadsheet formulas to fetch data from other web services, using custom UI elements to edit form data, and more.

We present examples of using Wildcard to solve real world problems, and explain the design principles behind the prototype. In the future, we envision continuing to build Wildcard into a fully deployed system that makes the web into a more malleable medium.

ACM CCS 2012

- *General and reference* → *Computing standards, RFCs and guidelines*;
- **Applied computing** → **Publishing**;

**Keywords** end-user programming, software customization, web browser extensions

## The Art, Science, and Engineering of Programming

Perspective The Art of Programming

Area of Submission Programming environments, Visual and live programming



© Geoffrey Litt and Daniel Jackson  
This work is licensed under a “CC BY 4.0” license.  
Submitted to *The Art, Science, and Engineering of Programming*.

### 1 Introduction

In 2012, the travel site Airbnb removed the ability to sort listings by price. Users could still filter down to a price range, but could no longer view the cheapest listings first. Many users complained on online message boards that the change seemed hostile to users. “It’s so frustrating!..What is the logic behind not having this function?” said one user on the Airbnb support forum. Alas, the feature remains missing to this day.

This is a familiar situation in a world of web applications that are frequently updated without user consent. For most people, when web software does not quite meet their needs, their only recourse is to complain to the developers and hope someone listens. If they know how to program in Javascript, perhaps they can implement a user script or a browser extension to patch the issue, but most people do not have these programming skills. While many have become accustomed to this status quo, we see it as a waste of the openness of the Web platform and the general pliability of software. In *Personal Dynamic Media*, Alan Kay envisioned personal computing as a medium that let a user “mold and channel its power to his own needs,” but today’s software is far from this vision.

In this paper, we introduce Wildcard, a browser extension that aims to make software more malleable by enabling users to tweak web applications without programming. Wildcard adds a panel to the bottom of a web page that shows a structured table view of the main data in the page. The table maintains a bidirectional connection to the original page—when the user manipulates the table, the original page gets modified, and vice versa.

In Wildcard, a user can sort Airbnb listings with just one intuitive click on a table header, with no programming required. Beyond sorting and filtering data, Wildcard also supports accessing third party APIs, performing small computations, recording private user annotations, using alternate UI widgets, and other useful changes. While Wildcard does not support all changes someone might want to make to a website, it makes broad subset of changes easily accessible to end users.

Under the hood, the implementation is straightforward, because a programmer must manually write an adapter for each individual website, which uses web scraping techniques to map the web page to the table. *todo: mention the other bits of extension code too* While programming is required for part of the process, this is still very different from traditional browser extensions—instead of the programmer defining a narrow use case, the end user is able to make many different changes on top of a single site-specific adapter. Programmers can extend Wildcard with plugins which provide various bits of functionality including connecting it to specific websites and web APIs, but the end user never needs to do any traditional programming.

*note the current prototype stage*

In this paper, we present examples of using Wildcard to solve real world problems, and explain the design principles behind the prototype:

*todo: bullet point the design principles here*

In the future, we envision building Wildcard into a fully deployed system that makes the web into a more malleable medium. *todo: this needs something more*

## **2 Demos**

To get a sense of how it feels to use Wildcard, let's see an example of using it to help with booking a trip using the travel sites Airbnb and Expedia.

### **2.1 Sorting and filtering**

We start by opening up the Airbnb search listings page to look for a place to stay. As mentioned before, this page annoyingly doesn't let us sort by price, so we'll use Wildcard to fix that. First, we open up the Wildcard panel, which shows a table corresponding to the search results in the page. As we click around in the table, the corresponding row in the page is highlighted so we can see the connection between the views.

To sort the page in ascending order by price, all we need to do is click on a table header to sort the table, and the original page gets sorted in the same order. We can also filter to only listings with a rating above 4.5 using a filtering UI on the column header. (Filtering by rating is another feature not offered in the Airbnb site.)

Note how after finishing the sorting and filtering, we can close the table view and continue using the website in its original design. The Wildcard interface is better for flexibly manipulating the underlying data, but the original site will usually offer a more polished and pleasant experience for viewing the data after it's in the desired form.

### **2.2 Row actions**

Most websites that show tables of data also offer various actions that can be taken on a row in the table, like adding an item to a shopping cart. Wildcard has the ability to make these actions available in the data table, passed through by the site-specific adapter. The main advantage this provides to users is the ability to easily perform an action in bulk across multiple rows.

For example, it's tedious on Airbnb to click on listings one by one to add them to a list of favorites, or open the listings in a new tab. Using Wildcard, we can just select multiple rows, right click, and then choose an action from the context menu to apply to all the rows.

Within the site adapter, each action is implemented as a web automation that can do anything available in Javascript running in the context of the page: clicking on buttons in the UI, launching AJAX requests, navigating to a new page, etc.

### **2.3 User annotations**

It's often useful to annotate a web page with notes. Users can use Wildcard to annotate a page by adding data into a new column, which is then shown in context of the original page.

Here, we use this feature to jot down notes on pros and cons of various listings:

## Wildcard: End user modification of web applications through a data table

When we come back later to the site, the annotations will still be visible. To support this, the site adapter saves the user annotations to browser local storage for persistence, and associates each table row with a stable identifier from the original site.

### 2.4 Computation with formulas

The demos so far have shown small, straightforward tweaks that provide useful conveniences while requiring very little effort. But Wildcard also supports adding more sophisticated functionality through a formula system.

When traveling without a car, it's nice to evaluate potential places to stay based on how walkable the surroundings are. There's an online service called Walkscore that can rate the walkability of any location on a 1-100 scale. It would take way too much work to manually cross-reference the scores with Airbnb locations, but we can use Wildcard formulas to automatically integrate Walkscore values into the page.

Wildcard includes a formula that uses the Walkscore API to fetch the score for any latitude and longitude. When we call the formula, it fetches the associated score and populates it into the page. By copy pasting the formula into all the rows on the page, we can add Walkscore data to all the search listings:

Programmers can extend Wildcard with new formulas, which are just Javascript functions that take table data as input and return new data. Formulas can access external APIs to fetch data, but cannot have side effects that directly manipulate the page. Instead, they return data, which the user can choose to add into the page by showing a column of data in the page.

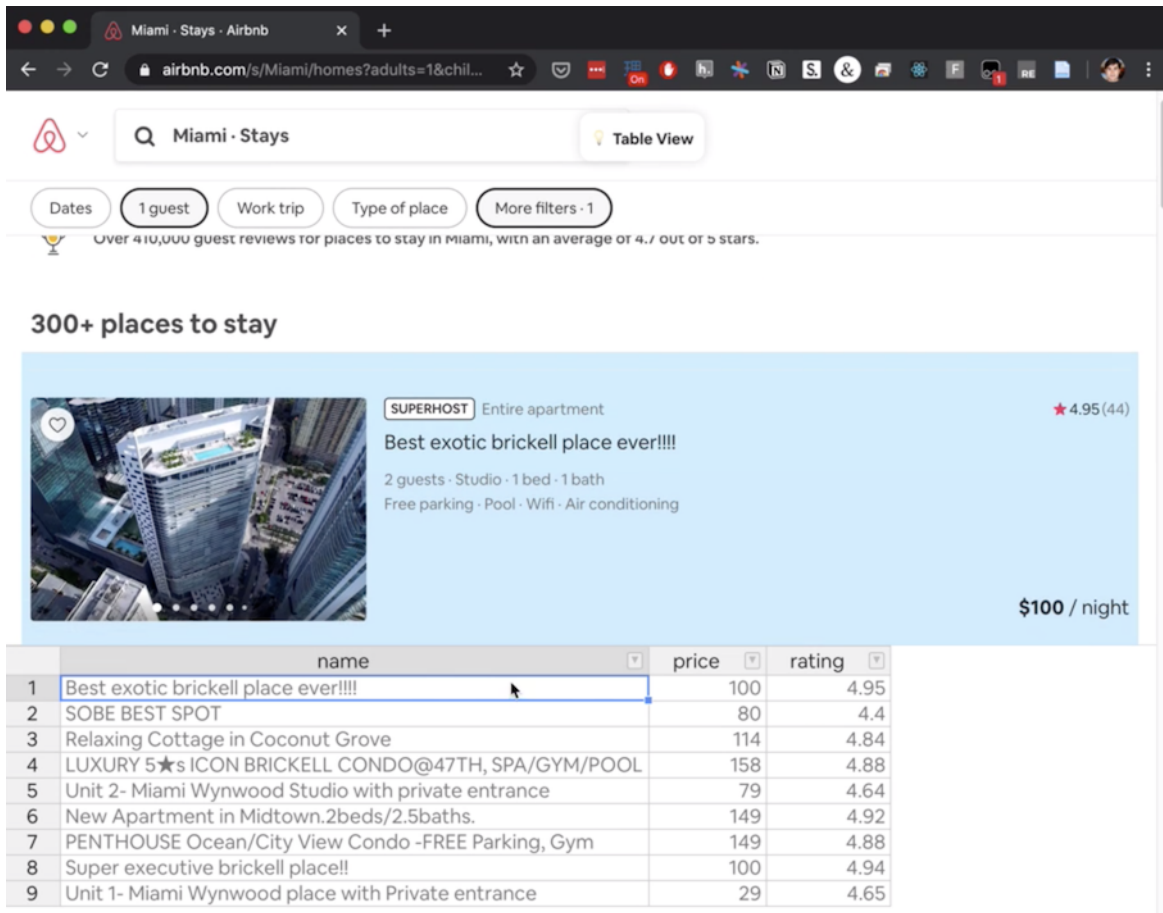
*todo: demo of composing formulas (requires a real formula system...)*

### 2.5 Custom UI elements

It might seem that Wildcard is only useful on websites that display lists of tabular data like search results. But in fact, the table metaphor is flexible enough to represent many types of data. For example, a flight search form on Expedia.com can be represented as a table with a single row:

This alone doesn't provide additional value on top of the original form, but it becomes useful when combined with two other features of Wildcard. First, Wildcard offers writable cells, where edits in either the table or the original site propagate in both directions. Second, Wildcard offers UI widgets that allow users to graphically edit the value of a cell.

Here's an example of using those features to help with filling in a form. When filling in dates for a flight search, it's important to correctly remember the planned dates for the trip. This often requires opening up a separate calendar application to look up the dates, and manually entering the dates into the form. In Wildcard, we can do this without manual copying, by editing the date cell directly using a datepicker widget that has access to our calendar information. Notice how the dates in the original form update when we update the table cells.



Over 410,000 guest reviews for places to stay in Miami, with an average of 4.7 out of 5 stars.

**300+ places to stay**

**SUPERHOST** Entire apartment  
 Best exotic brickell place ever!!!!  
 2 guests · Studio · 1 bed · 1 bath  
 Free parking · Pool · Wifi · Air conditioning  
 \$100 / night

	name	price	rating
1	Best exotic brickell place ever!!!!	100	4.95
2	SOBE BEST SPOT	80	4.4
3	Relaxing Cottage in Coconut Grove	114	4.84
4	LUXURY 5★s ICON BRICKELL CONDO@47TH, SPA/GYM/POOL	158	4.88
5	Unit 2- Miami Wynwood Studio with private entrance	79	4.64
6	New Apartment in Midtown.2beds/2.5baths.	149	4.92
7	PENTHOUSE Ocean/City View Condo -FREE Parking, Gym	149	4.88
8	Super executive brickell place!!	100	4.94
9	Unit 1- Miami Wynwood place with Private entrance	29	4.65

■ **Figure 1** Opening a table corresponding to search results on Airbnb

Custom UIs enable people to use a consistent UI to enter common types of data across the web. They also allow a user to access their own private data as part of a web interface, without needing to expose that private data to the website.

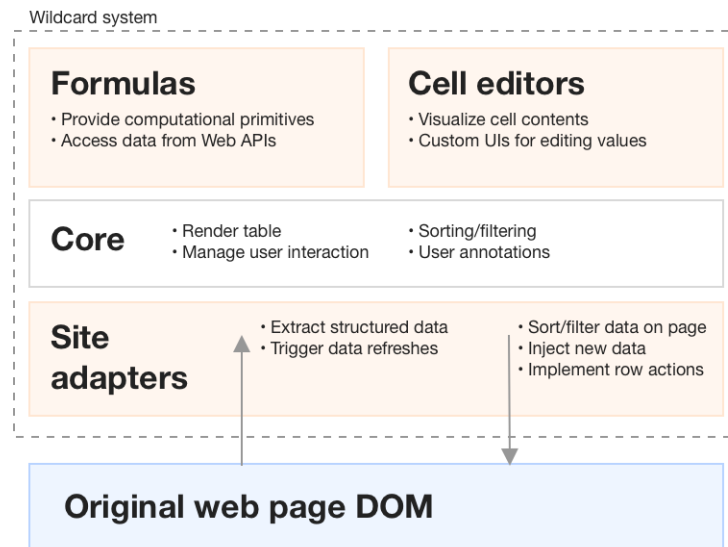
The PDF version of the output will go here.

Probably make one giant full-page figure that explains the whole thing?

For example, in Fig. 1 we open up a table view that corresponds to search results on the Airbnb travel site.

Overall, we think an interactive data table is a natural computation model that presents a surprisingly large range of possibilities for end user modification of websites. While we've presented just a small sampling of use cases here to concretely illustrate some of these possibilities, we plan to continue developing site adapters, formulas, and UI elements to explore more use cases, and to eventually publicly release the tool to allow real end users to discover their own uses.

## Wildcard: End user modification of web applications through a data table



■ **Figure 2** The architecture of the Wildcard system

### 3 System Architecture

Wildcard is written in Typescript, and is injected into pages using the Tampermonkey userscript manager (although in the future we plan to deploy it as a standalone browser extension.)

In order to maximize extensibility by other programmers, Wildcard is implemented as a small core program along with several types of plugins: site adapters, formulas, and cell renderers/editors. The core contains functionality for displaying the data table and handling user interactions, and the table implementation is built using the Handsontable Javascript library.

Site adapters specify the bidirectional connection between the web page and its structured data representation.

For extracting data from the page and getting it into structured form, Wildcard provides ways to concisely and declaratively express scraping logic. For example, here is a code snippet for extracting the name of an Airbnb listing. It specifies the field name and some metadata properties, and then a function that extracts the name div from a row div.

```
{
  fieldName: "name",
  readOnly: true,
  type: "text",
  el: (row) => row.querySelector(`${titleClass}`),
}
```

The site adapter also needs to support the reverse direction: sending updates from the table to the original page. Most DOM manipulation is not performed directly by

the site adapter. To support sorting and filtering, the adapter specifies how to find the divs representing data rows, and the core platform sorts and filters the DOM to reflect the table state. User annotations are handled similarly. Finally, site adapters can also optionally implement site-specific actions (like favoriting in Airbnb) in normal Javascript code that controls the page.

So far we have implemented *X (fill in latest number)* site adapters built into Wildcard. We plan to implement more in the near future to continue to test the limits of the API, and to encourage third party programmers to build, maintain, and distribute adapters for popular sites.

Formulas and cell editors are also implemented as separate modules, so that additional ones could be contributed by third party programmers.

## **4 Design principles**

The design of Wildcard is grounded in several principles, informed by prior work and our own experimentation. We think these principles can also more generally inform the design of end user programming, particularly systems enabling users to modify GUI apps.

### **4.1 Decouple UI from data**

Most software does not allow users to choose their own UI elements, even for common data types. If a website provides a datepicker widget, you have no ability to provide your own replacement datepicker, with your preferred design or with privileged access to your private calendar data. This forces users to learn many different interfaces of varying quality for similar tasks. Some websites have APIs to allow users to access the data underlying the UI, but in addition to requiring programming, these are heavyweight tools more fit for batch exports or building entire new clients than for casual UI modification.

In Wildcard, a user gets access to a view of the underlying data in the page, and can choose their own interfaces to view and modify the data. The Expedia datepicker demo showed one example of how this can be useful, but we also envision creating other widgets for visualizing and editing data. Some examples would be showing geographic data in a custom map that includes the user's own annotations, or editing a blog post in a rich text editor of the user's choice.

One benefit of decoupling data from interfaces is improved UI quality. When UI widgets can compete on their merits rather than based on network effects from the data they have access to, it creates much stronger competition at the interface level. For example, there is competition among email clients (which consume an open protocol), but not among Facebook or Twitter clients. This benefit relates to the SOLID project led by Tim Berners-Lee [5], which envisions user-controlled data as a mechanism for decoupling data from interfaces, e.g. giving users a choice of which client to use to consume a given social media feed. Wildcard has overlapping goals

## Wildcard: End user modification of web applications through a data table

with SOLID, but does not require decentralized user control of data—the data can remain on a centralized server, as long as the interface can be tweaked by end users.

Another benefit of decoupling data from UI is that it becomes possible to use the same consistent interface across many applications. For example, many programmers become deeply familiar with one text editor and use it for many different kinds of tasks, even as an interactive input mechanism in the shell (e.g. for editing git commit messages). The ability to generically reuse the text editor in many contexts makes it worth investing time in mastering the tool. Beaudouin-Lafon and Mackay refer to this ability to use a UI tool in many contexts as *polymorphic* interaction [3], noting that it is a useful technique for keeping interfaces simple while increasing their power. diSessa also [11] notes that there is a connection between polymorphism and the idea of literacy in a medium: textual literacy rests on a single rich medium of writing which can be adapted to many different genres and uses.

*Note: Maybe could relate this section to Concept Reuse?*

### 4.2 Expose structure

In *Changing Minds* [11], Andrea diSessa critiques the design of modern software with a story about a hypothetical “nightmare bike.” Each gear on the nightmare bike is labeled not with a number, but with an icon describing its intended use: smooth pavement uphill, smooth pavement downhill, gravel, etc. This might seem more “user-friendly” than numbered gears, but in fact, it makes it harder to operate the bike. A numerical sequence allows the user to develop intuition for the structure of the system, but isolated modes provide a superficial understanding with no grounding in structure. This understanding might be sufficient for the most common cases but breaks down in unfamiliar situations. If someone needs to go uphill on gravel, do they need to try every mode at random?

Many modern software designs fall into this trap, teaching users to use isolated modes rather than coherent structure, and the problem gets far worse when operating across multiple applications. Unlike the UNIX philosophy of small tools interoperating through shared abstractions, in modern computing each application is in its own silo of data and functionality.

Wildcard helps people understand and modify the behavior of applications through the lens of a consistent abstraction: a data table. This abstraction strikes a balance between being both simple and generic. A data table is simpler than the DOM tree that describes the details of the UI, but is also generic enough to describe the essence of many different kinds of applications.

Creating a structured abstraction to represent a web page is a deliberate choice, and is not the only way to enable users to modify websites without directly accessing the DOM. Systems like Chickenfoot [6] and CoScripter [19] allow users to create scripts in an informal language and then perform fuzzy pattern matching to find elements in the DOM. For example, to find a button after a textbox in Chickenfoot, the user could type `click(find(“button just after textbox”))`. These designs allow for expressing a wide range of operations, but they don’t explicitly indicate what operations are possible—the user can only see the original page and imagine the possibilities. In



contrast, Wildcard provides affordances that clearly suggest the availability of certain actions (e.g. sorting, editing a cell, adding a column with a derived value), especially to users who are familiar with spreadsheets. In addition to giving users more certainty about whether a modification is possible, these affordances might give users new ideas for things to try. Just as graphical interfaces better communicate the space of possible actions than command line interfaces, Wildcard aims to clearly communicate the space of possible modifications.

### 4.3 Direct manipulation of an alternate representation

In Wildcard, users manipulate an alternate representation of a web page. The interaction with the data table is direct like using a spreadsheet, but the interaction with the page is indirectly mediated through the table.

We considered other approaches where the user would interact more directly with the original UI, e.g. injecting sort controls into the page, but decided that the table view had advantages that justified the cost of adding a layer of indirection:

- *Consistency*: Even across different websites, the table view always has the same layout, making it easier to learn to use.
- *Affordances*: The table view suggests possible actions like adding a new column, which are challenging to suggest in the context of the original page.
- *Blank slate for UI*: When a custom UI element is used to manipulate a cell in the data table, there are no conflicts with the existing interface of the site.

The main challenge of making this design successful is maintaining a close mapping in the user's mind between the new representation and the original page (*note: cite Norman? Cognitive Dimensions 'closeness of mapping'?*). Wildcard provides live visual cues as the user navigates the data table, similar to the highlighting provided by browser developer tools to indicate the mapping between HTML and the original page. In practice in our own usage, we have found that this live highlighting is sufficient to make it clear how the two representations map to each other.

### 4.4 Encourage casual tweaking

	<i>Casual</i>	<i>Not casual</i>
<i>End user friendly</i>	<b>Wildcard</b>	IFTTT
<i>Requires programming</i>	browser dev tools	editing open source desktop applications

We can evaluate a system for modifying software along two dimensions. First, the technical capability required of the user: is programming knowledge needed? Second, the level of effort required: how far out of their way the user must go to make a change? Can they casually make a small change, or do they need to make a larger project out of it? These dimensions are not orthogonal, but they are distinct. For example, setting up a workflow trigger in an end user programming system like IFTTT does not require much technical skill, but it does require leaving the user's

## **Wildcard: End user modification of web applications through a data table**

normal software and entering a separate environment. On the other hand, running a Javascript snippet in the browser console requires programming skills, but can be done immediately and casually in the flow of using a website.

In addition to requiring no programming skills, Wildcard also aims to support frequent, small modifications. The Wildcard table appears in the course of normal web browsing, to ensure that the tools for modification are close at hand while using the original software. Ink and Switch refers to this property as having an “in-place toolchain” [16].

We also try to make simple changes possible with particularly low effort, like being able to sort a table in a single click. This property is inspired by spreadsheets, which can be useful even to someone who has learned only a small part of their functionality. In contrast, many traditional programming systems require someone to learn many complex concepts just to perform a simple task (e.g., needing to learn what public static void main means to write a Hello World program in Java).

### **4.5 First party cooperation optional**

The Web is an unusually extensible platform. On many other platforms (e.g. smart-phone operating systems), software is locked down unless first-party developers explicitly provide hooks for plugins and interoperability, but on the Web, all client-side code is available for browser extensions to modify. Application authors can use practices that make it easier to modify their apps (e.g. clean semantic markup), or more difficult (e.g. code obfuscation), but the default state is openness. This gives extensions freedom to modify applications in creative ways that the original developers did not plan for. At the most extreme, this can even include modifications like ad blocking that are hostile to the desires of the original developers (a notion Cory Doctorow calls adversarial interoperability), but there are also many other changes which are not actively hostile, merely falling outside the range of possibilities originally imagined by the developer.

Wildcard takes advantage of this openness, and does not depend on cooperation from first-party website developers. Any programmer can add support for any website to Wildcard by building a third party adapter. This design decision acknowledges the pragmatic need to interoperate with current websites, but we hope that eventually first party website developers will build in Wildcard support to their applications, since this would reduce the burden of maintaining adapters and make Wildcard plugins more stable.

Implementing the Wildcard adapter API could help developers by allowing users to fix some of their own issues, particularly idiosyncratic use cases that the first party developer would never plan to prioritize. Supporting Wildcard could be straightforward in a typical client-side application that already has access to a structured version of the data in the page. And while some developers might hesitate to promote extensibility in their clients to avoid unwanted changes, the most common problem of users blocking ads is already ground well trod by existing browser extensions. There is also precedent for first parties implementing an official client extension API in response to user demand: for several years, Google maintained an official extension API in

Gmail for Greasemonkey scripts to use. (Incidentally, since then, third parties have continued to maintain Gmail extension APIs used by many Gmail browser extensions [22, 23], illustrating the value of collaboratively maintaining third party adapters.)

#### 4.6 Collaboration over automation

The architecture of Wildcard requires site adapters to abstract the low level representation of a web page into a higher-level, structured representation that is easier for end users to work with. Many web scraping research systems have either automated this process entirely using heuristics, or created interactive workflows for end users to specify the translation. In Wildcard, we take a different approach: programmers manually code site adapters.

This decision is partially a pragmatic one—we haven’t had time to incorporate sophisticated automation yet, and we may choose to do so in the future. But it is also a principled decision. Automated approaches tend to work much of the time but not all of the time [6] [20]. In contrast, human programmers have been able to maintain very stable scraping libraries, even for complex sites like Gmail [22, 23]. We are willing to trade off lower site coverage in return for more reliable results on the sites that are covered, because this builds confidence for users about where they can expect Wildcard to work.

We see this architecture as a form of collaboration between programmers and end users. In the wild, spreadsheets are built collaboratively by users with different levels of technical expertise: some users write simple formulas, while others help write more complex formulas or even macros [21]. Even though some parts of the collaboration require more expertise, even users with little expertise are still empowered to perform much of the work themselves, which differs radically from custom software development where coding expertise is table stakes for making any contributions. Similarly, we envision multiple layers of Wildcard usage depending on technical expertise:

- only using sorting/filtering/annotation, with no computation of any kind
- writing simple formulas to fetch new data
- composing complex formulas that perform more advanced computation
- using Javascript to create or maintain site adapters, formulas, or cell editors

## 5 Related work

Many projects have explored end user customization of web applications, but it has not become a common activity in the wild. We suspect that so far no system has found the right combination of utility, ease of use, and robustness required for widespread adoption. Wildcard aims to build on lessons learned from this body of work, employing a novel approach of using a familiar spreadsheet table to go about the task of modifying the UI of an existing web application. Here we describe some of the areas of related work, and how Wildcard builds on them.

## **Wildcard: End user modification of web applications through a data table**

### **5.1 Malleable software**

In the broadest sense, Wildcard is inspired by systems aiming to make software into a dynamic medium where end users frequently create and modify software to meet their own needs, rather than accepting pre-built applications built by programmers. These systems include Smalltalk [17], Hypercard [15], Boxer [10], Webstrates [18], and Dynamicland [24]. (The project's name Wildcard comes from the internal pre-release name for Hypercard, which promoted software modification by end users and was a precursor to the modern Web.)

These projects generally require building new software from scratch for that environment. This provides a great deal of flexibility, but also means they are not compatible with existing software and require wholesale adoption. In contrast, Wildcard takes a pragmatic approach of enabling people to tweak the software they already use.

### **5.2 Web augmentation**

Wildcard's goals are closely shared with other browser extensions which provide panels to augment and modify a website in the context of using the site.

Sifter [13] enables users to sort and filter lists of data on web pages by converting the data into a structured format, providing a result similar to Wildcard's sort and filter functionality. The mechanism is different, though: Sifter uses a combination of automated heuristics and interactive user feedback to extract data, whereas Wildcard currently relies on programmers creating wrappers for extracting structured data, probably leading to higher quality extraction but on fewer sites. Also, Wildcard aims for much broader functionality than just sorting and filtering, and shows the structured table view of the data to facilitate these other interactions, whereas Sifter just shows sort controls and never reveals the underlying data table.

"Sloppy programming" [20] tools like Chickenfoot [6] and Coscripter [19] enable users to create scripts that perform actions like filling in text boxes and clicking buttons, without directly interacting with the DOM. Users express the desired page elements in natural, informal terms (e.g. writing "the username box" to represent the textbook closest to the label "username"), and then using heuristics to determine which elements most likely match the user's intent. This approach allows for expressing a wide variety of commands with minimal training, but it also has downsides. It is difficult to know whether a command will consistently work over time (in addition to changes to the website, changes to the heuristics can also cause problems), and it is not easy for users to discover the space of possible commands.

Wildcard offers a sharp contrast to sloppy programming, instead choosing to expose a high degree of structure through the familiar spreadsheet table, which offers the reverse set of tradeoffs: more robust behavior, but in a narrower domain. Wildcard offers more consistency (e.g., clicking a sort header will always work correctly as long as the site adapter is maintained) and offers clearer affordances for what types of actions are possible (or, crucially, what actions are *not* possible, which is very useful for users to know). On the other hand, Wildcard cannot offer coverage of all websites, and has a narrower range of possible actions than sloppy tools. Still, we hope that

with enough site adapters and formulas, these downsides can be mitigated, and that the benefits of structure outweigh these costs.

### 5.3 Spreadsheet-based app builders

It is a powerful realization to notice that a spreadsheet can serve as an end-user-friendly backing data store and computation layer for an interactive web application. Research projects like Quilt [4], Gneiss [7], and Marmite [25], as well as commercial tools like Airtable Blocks [1] and Glide [9] allow users to view data in a spreadsheet table, compute over the data using formulas, and connect the table to a GUI. Because many users are already familiar with using spreadsheets, this way of creating applications tends to be much easier than traditional software methods; for example, in a user study of Quilt, many people were able to create applications in under 10 minutes, even if they expected it would take them many hours.

Wildcard builds on this powerful idea, but applies it to modifying existing applications, rather than building new applications from scratch. For many people, we suspect that tweaking existing applications provides more motivation as a starting point for programming than creating a new application from scratch.

An important design decision for tools in this space is how far to deviate from traditional spreadsheets, and in what ways. Quilt and Glide use existing web spreadsheet software as a backend, providing maximum familiarity and compatibility with existing spreadsheets, but other projects branch out in various ways. Gneiss has its own spreadsheet with additional features useful for building GUIs. Marmite provides a live data view that resembles a spreadsheet, but programming is actually done using a separate data flow pane rather than spreadsheet formulas, which led to some confusion in a user study among users who expected behavior more similar to spreadsheets [25]. Airtable offers a relational database with formula support in a spreadsheet-style direct manipulation interface, a combination which has found commercial success. Wildcard’s table is most similar to Airtable, aiming to offer the structure of a relational table, but to keep the notion of pure reactive formulas from spreadsheets rather than introducing a separate programming model outside of the table.

Quilt also raised the hypothetical idea of “web worksheets” in spreadsheets: small HTML UIs inside of a spreadsheet table to help users edit data or formulas in a more domain-specific interface. This is similar to Wildcard’s notion of cell editors.

### 5.4 Web scraping / data extraction

Web scraping tools focus on extracting structured data out of unstructured web pages. Web scraping is closely related to the implementation of Wildcard, but has very different goals: web scraping extracts static data for processing in another environment, whereas Wildcard supports modification of the original page by maintaining a bidirectional connection between the extracted data and the page.

Web scraping tools differ in how much structure they attempt to map onto the data. Some tools like Rousillon [8] extract data in a minimally structured relational format; other tools like Piggy Bank [14] or Thresher [12] more ambitiously map the data to

## Wildcard: End user modification of web applications through a data table

a Semantic Web schema. In Wildcard, we chose to avoid rich schemas, in order to reduce the work associated with creating a site adapter.

### 6 Limitations and future work

Wildcard is still in early development; there are still many limitations to resolve and open questions to explore.

The most important question is whether the combination of features provided by Wildcard can actually be combined in enough useful ways to make the system worth using in practice. We plan to test this by first continuing to privately test the system with our own needs, and eventually by shipping the system publicly, once the API is stable enough and we have developed enough site adapters and formulas to support a critical mass of sites and use cases.

Here are a few of the most pressing limitations in the current system:

- Wildcard only extracts data on the page, which means that subsequent pages of results in paginated lists are not included in the table. (Sifter explains [13] techniques that might help get around this.)
- There is no mechanism for end users to express imperative workflows (e.g. “click this button, and then...”); they can only write formulas that return data and then inject the resulting data into the page. While this makes the system simpler, it also may exclude many valuable use cases.
- Wildcard’s data model expresses a single table at a time, without any notion of relationships between tables. A UI like SIEUFERD [2] might help to visualize joins between tables.
- Only allowing site adapters to be manually coded means that the number of supported sites is limited. We may explore also building wrappers using automated heuristics.

### 7 Conclusion

We live in a time of pessimism about the effects of the internet and software. At times it seems like there is no way out of the mess, but we think that one promising solution is to enable more people to have deeper control over the software they use every day. The Web offers an open foundation that browser extensions have capitalized on, but even this has only empowered users within the limited confines of the ideas that programmers have been able to think of.

In this paper, we have presented Wildcard, a general-purpose browser extension that gives end users more freedom to mold web applications to meet their needs. To receive future updates on Wildcard and notifications about a public release, sign up for the email newsletter.

### Acknowledgements

## References

- [1] *Airtable: Organize Anything You Can Imagine*. <https://airtable.com>.
- [2] Eirik Bakke and David R. Karger. “Expressive Query Construction through Direct Manipulation of Nested Relational Results”. en. In: *Proceedings of the 2016 International Conference on Management of Data - SIGMOD ’16*. San Francisco, California, USA: ACM Press, 2016, pages 1377–1392. ISBN: 978-1-4503-3531-7. DOI: 10.1145/2882903.2915210.
- [3] Michel Beaudouin-Lafon and Wendy E. Mackay. “Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces”. In: *Proceedings of the Working Conference on Advanced Visual Interfaces*. AVI ’00. Palermo, Italy: ACM, 2000, pages 102–109. ISBN: 978-1-58113-252-6. DOI: 10.1145/345513.345267.
- [4] Edward Benson, Amy X. Zhang, and David R. Karger. “Spreadsheet Driven Web Applications”. en. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST ’14*. Honolulu, Hawaii, USA: ACM Press, 2014, pages 97–106. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647387.
- [5] Tim Berners-Lee. *One Small Step for the Web*. . . en. [https://medium.com/@timberners\\_lee/one-small-step-for-the-web-87f92217d085](https://medium.com/@timberners_lee/one-small-step-for-the-web-87f92217d085). Sept. 2018.
- [6] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. “Automation and Customization of Rendered Web Pages”. en. In: *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology - UIST ’05*. Seattle, WA, USA: ACM Press, 2005, page 163. ISBN: 978-1-59593-271-6. DOI: 10.1145/1095034.1095062.
- [7] Kerry Shih-Ping Chang and Brad A. Myers. “Creating Interactive Web Data Applications with Spreadsheets”. en. In: *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology - UIST ’14*. Honolulu, Hawaii, USA: ACM Press, 2014, pages 87–96. ISBN: 978-1-4503-3069-5. DOI: 10.1145/2642918.2647371.
- [8] Sarah E. Chasins, Maria Mueller, and Rastislav Bodik. “Rousillon: Scraping Distributed Hierarchical Web Data”. en. In: *The 31st Annual ACM Symposium on User Interface Software and Technology - UIST ’18*. Berlin, Germany: ACM Press, 2018, pages 963–975. ISBN: 978-1-4503-5948-1. DOI: 10.1145/3242587.3242661.
- [9] *Create an App from a Google Sheet in Minutes · Glide*. en. <https://www.glideapps.com/>.
- [10] A. A diSessa and H. Abelson. “Boxer: A Reconstructible Computational Medium”. en. In: *Communications of the ACM* 29.9 (Sept. 1986), pages 859–868. ISSN: 00010782. DOI: 10.1145/6592.6595.
- [11] Andrea A. diSessa. *Changing Minds: Computers, Learning, and Literacy*. Cambridge, MA, USA: MIT Press, 2000.
- [12] Andrew Hogue and David Karger. “Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web”. en. In: *Proceedings of the 14th International Conference on World Wide Web - WWW ’05*. Chiba, Japan: ACM Press, 2005, page 86. ISBN: 978-1-59593-046-0. DOI: 10.1145/1060745.1060762.

## Wildcard: End user modification of web applications through a data table

- [13] David F. Huynh, Robert C. Miller, and David R. Karger. “Enabling Web Browsers to Augment Web Sites’ Filtering and Sorting Functionalities”. en. In: *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology - UIST ’06*. Montreux, Switzerland: ACM Press, 2006, page 125. ISBN: 978-1-59593-313-3. DOI: 10.1145/1166253.1166274.
- [14] David Huynh, Stefano Mazzocchi, and David Karger. “Piggy Bank: Experience the Semantic Web Inside Your Web Browser”. en. In: *The Semantic Web – ISWC 2005*. Edited by Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pages 413–430. ISBN: 978-3-540-32082-1. DOI: 10.1007/11574620\_31.
- [15] Hypercard. “HyperCard”. en. In: *Wikipedia* (Dec. 2019). Page Version ID: 931376685.
- [16] Ink and Switch. *End-User Programming*. en-US. Mar. 2019.
- [17] A. Kay and A. Goldberg. “Personal Dynamic Media”. In: *Computer* 10.3 (Mar. 1977), pages 31–41. ISSN: 1558-0814. DOI: 10.1109/C-M.1977.217672.
- [18] Clemens N. Klokmoose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. “Webstrates: Shareable Dynamic Media”. en. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST ’15*. Daegu, Kyungpook, Republic of Korea: ACM Press, 2015, pages 280–290. ISBN: 978-1-4503-3779-3. DOI: 10.1145/2807442.2807446.
- [19] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. “CoScripter: Automating & Sharing How-to Knowledge in the Enterprise”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’08. Florence, Italy: ACM, 2008, pages 1719–1728. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357323.
- [20] Greg Little, Robert C. Miller, Victoria H. Chou, Michael Bernstein, Tessa Lau, and Allen Cypher. “Sloppy Programming”. en. In: *No Code Required*. Elsevier, 2010, pages 289–307. ISBN: 978-0-12-381541-5. DOI: 10.1016/B978-0-12-381541-5.00015-8.
- [21] Bonnie A. Nardi and James R. Miller. “An Ethnographic Study of Distributed Problem Solving in Spreadsheet Development”. In: ACM Press, 1990, pages 197–208.
- [22] Streak. *InboxSDK*. <https://www.inboxsdk.com/>.
- [23] Kartik Talwar. *Gmail.Js*. en. <https://github.com/KartikTalwar/gmail.js>. 2019.
- [24] Bret Victor. *Dynamicland*. <https://dynamicland.org/>.
- [25] Jeffrey Wong and Jason I. Hong. “Making Mashups with Marmite: Towards End-User Programming for the Web”. en. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI ’07*. San Jose, California, USA: ACM Press, 2007, pages 1435–1444. ISBN: 978-1-59593-593-9. DOI: 10.1145/1240624.1240842.



### About the authors

**Geoffrey Litt** is a PhD student at MIT, researching programming tools for end users and developers.

**Daniel Jackson** is a professor in the Department of Electrical Engineering and Computer Science at MIT, associate director of CSAIL, and a MacVicar Fellow. He leads the Software Design Group.