Declarative Setup-free Web Application Prototyping Combining Local and Cloud Datastores

Filip Kis
KTH Royal Institute of Technology
Stockholm, Sweden
Email: fkis@kth.se

Cristian Bogdan
KTH Royal Institute of Technology
Stockholm, Sweden
Email: cristi@kth.se

Abstract—The trend of “open data” coupled with the recent advancement in web development technologies and the proliferation of JavaScript frameworks has helped popularize programming of interactive web applications. Still, some of the common features of today’s web applications that access data from own data stores or from web services require a complex setup or a significant amount of programming knowledge, and thus make it hard for developers to quickly prototype applications and iterate on solutions. Therefore, we propose Endev, a declarative framework for prototyping applications that use cloud data storage or web service data. By not needing to write any JavaScript code or set up any servers, Endev provides a low learning threshold. We show that Endev is perceived useful and easy to use through a study with 15 developers.

I. INTRODUCTION

There is an increasing interest from developers and end-users to access the data available on the Internet and use it for their personal or professional interests [24], [30]. This has led to the “open data” trend where more and more institutions and on-line services offer programmable access to their data through web services. According to ProgrammableWeb1 there are 14795 web service APIs as of March 2016 and the majority are made available through simple REST interfaces [19] which makes the data easily accessible.

Furthermore, the proliferation of open-source web development frameworks (e.g. Bootstrap, Ember, Angular, React), which support designing more complex layout and reusing components for advanced features, has made building web applications faster. However, adding interactivity and populating HTML user interfaces with data coming from web services is still a challenge [37]. Additionally, AJAX technology used for accessing web service data asynchronously is complex to write and relies on callbacks which produce error-prone spaghetti code [26], [27].

In the past decade researchers have recognized the issues of using web service data and proposed solutions in the form of mashup tools [33], [36]. They allow users to extract data from various sources and combine them in one view without having to write code. However, these tools are not widely used for building interactive applications as they provide only a limited control over displaying the data. An additional limitation of mashup solutions is that they typically work with data composition only. Developers and end-users, however, often want to also store their own data [34] and it was estimated that by 2012 over 55 million people in the US would “manipulate data with either spreadsheets or databases at work” [32]. However, beyond spreadsheets, setting up and using a database requires advanced setup and programming knowledge.

Our experience in teaching Interaction Programming to students with various backgrounds and organizing series of weekend-long “open data hackathons” revealed that the participants had the needed knowledge and skills to build graphically attractive static HTML prototypes, but struggled when these prototypes were required to work with real data. This is in line with research results showing that designers, and end-users in general, are skilled in HTML authoring, but desire the possibility to work with data [11], [28], [31].

To address these challenges we propose the following requirements for future tools for prototyping web applications working with data:

Simple data access – the tools need to provide easy-to-use abstractions over data access to remove the need for error-prone boilerplate code.

1http://www.programmableweb.com/
Control over UI design – in order to build creative and usable, and not just functional, applications the tools should have higher ceiling when it comes to design of interface and interaction.

Setup-free – to enable rapid prototyping and fast iterations there should be a low threshold when it comes to setting up the tools.

This paper introduces Endevelopment, a JavaScript framework that was built with those requirements in mind. Endevelopment supports building data-intensive web applications without the need of writing JavaScript code (as shown in Figure 1). Using Endevelopment, developers and end-users can build interactive web applications to create their own data or to leverage on web service data just by annotating HTML files. The main contributions of Endevelopment and this paper are:

Declarative annotations that use SQL-like syntax for binding to various data sources, removing the need for writing error-prone boiler plate code for data-access.

A client-side framework that provides on-the-fly data creation, three-way synchronization and real-time collaboration using existing frameworks in the spirit of modern web development practices.

Shareable prototypes as HTML and JavaScript that can be run in any browser and, since Endevelopment does not require a backend server, prototypes built with Endevelopment can easily be shared with developers/designers/end-users by just exchanging HTML, and possibly CSS, files (e.g. via email, Dropbox or code playgrounds like CodePen).

An evaluation conducted with 15 developers with the aim of investigating perceived Usefulness and perceived Ease of Use according to Technology Acceptance Model (TAM) [18].

The following section reviews related work on data-binding and web mashup tools. Next, we demonstrate the functionality of Endevelopment through two example applications and later break down its features and elaborate on the implementation details. We also introduce the study that was conducted for the purpose of evaluation and present its results. Finally, we discuss the results in the conclusion and present the future work.

II. RELATED WORK

A. Data-binding

Quilt [12] is a recent solution that provides HTML annotations to connect the interface to a spreadsheet that serves as the datastore. Quilt allows both data read and write and keeps the interface in synchronization (three-way binding) with the spreadsheet data. Any data sorting, formatting and computation need to be defined in the spreadsheets. The HTML annotations introduced by Quilt serve only for data-binding. Quilt takes advantage of the interface tree hierarchy to infer the relationships between parent and nested annotations. It uses only four different annotations and they have a different effect depending on what type of data they connect to. In contrast, Endevelopment provides more descriptive annotations, by utilizing SQL-like syntax, that are expected to result in a more understandable interface. Furthermore, while Quilt only offers access to data from spreadsheets, Endevelopment recognizes the necessity for allowing end-users to work with various data sources.

XFormsDB [35] is a declarative data binding solution that binds to server-side data. It is based on the XForms3, a W3C Recommendation, that was designed to be the next generation of HTML forms and used a Model-View-Controller approach. XFormsDB depends on having a complex server setup and supports only XML based databases, thus it is not ideal for quick prototyping. Furthermore, even though XForms are relatively old-standard (first version published in 2007), none of the major browsers currently natively supports it.

Industry client-side web development frameworks, such as Angular [7], Backbone [2], Ember [3] and Ractive [5], provide data-binding constraints that keep HTML elements automatically in synchronization with the application data values. In research, ConstraintJS [29] applies similar features as the industry frameworks, but extends their functionality to support finite-state machines. The drawback in terms of prototyping is that all of these solutions still require a certain amount of ordered statements in JavaScript code to provide the actual connection to the data source. In addition, they often require application of some form of Model-View-Controller or similar design pattern that end-users may not be familiar with. Endevelopment borrows the basic ideas of data-binding and templating, currently built on top of Angular, and expands them by the introduction of three-way, direct data-binding annotations that do not require JavaScript code and instead offer the end-user to define the data source directly in HTML annotations.

Our choice of SQL-like annotations stems from Makumba [22], which presented the idea of HTML template annotations that use SQL-like query annotations to link directly to the domain model objects. Makumba was easily understood and used by end-users [14] to maintain and expand a large production-level system. However, Makumba requires a complicated 3-tier backend setup, it does neither support web-service data nor modern web user interaction based on partial HTTP postback.

B. Mashups and web service data

Another solution based on spreadsheets as the datastore is Gneiss [17]. Unlike Quilt, Gneiss is a live programming mashup environment where, instead of using HTML annotations, the users can drag and drop widgets to the page and connect them with spreadsheet values. A key feature of Gneiss is the support for any REST web service returning JSON data which can be interactively combined in the spreadsheet before their data is used in the interface. The main drawback for Gneiss compared to Endevelopment is that the users are limited to working with UI widgets existing in the system, which significantly reduces the design possibilities.

In general, mashup tools provide end-users with the possibility to create web applications that combine data from various sources with little or no programming effort. This is typically achieved through direct-manipulation [9], [16], [25], programming-by-demonstration [20] or by allowing the users to record and automate their web behavior [4], [23]. However, all of these solutions only offer the possibility of combining

\footnote{Available for download and prototyping at http://www.endevjs.org/}

\footnote{https://www.w3.org/MarkUp/Forms/}
existing data sources and do not support user-owned data. Furthermore, the expressiveness of the interface and interaction is often limited by the predefined widgets supported by the tools. Endev purposefully takes a different approach in that it focuses on enabling the user to gather and modify the data easily, while leaving the layout and interaction in the users’ full control.

Data-flow language tools such as Yahoo! Pipes [8] support building mashups of web service data by direct-manipulation of input objects that can be linked together to produce output data. However, research has shown that data-flow programming is often difficult for end-users to understand [15]. The Yahoo Query Language (YQL) [1] technology addresses a similar goal of allowing the combination of various web service data, but instead of using a data-flow language and direct-manipulation, it uses a SQL-like syntax to support web service combination. The resulting data can be accessed via a REST API. Although both Yahoo! Pipes and YQL provide a uniform way of combining various web service data, there is still a threshold for end-users if they wish to use the resulting data in their own interfaces.

### III. Motivating Examples

We will introduce two examples to demonstrate the contribution of this paper. The first example demonstrates a use case where developers create and manipulate their own data. The second example demonstrates a case where external APIs are used to access data. The examples are first discussed in terms of how the respective functionality would be achieved without Endev, and then their Endev implementation is shown.

#### A. ToDo App

The first example is an application to create and manage a simple online to-do list, shown in Figure 1. The to-do list contains a text description of the task and a checkbox to mark the task completion. The list is shareable and it can be accessed and used by multiple users at the same time. For this to be achieved, the application requires data stored in a database and a real-time synchronization mechanism.

Implementing these features usually requires either setting up a database server or using some of the many versions of cloud-based services. Setting up a remote database server is not a trivial task and requires a hosting platform and administration knowledge, and thus is not an option for rapid prototyping or end-user development. Furthermore, the developer would also need to write and deploy backend code to be able to connect to the database.

An alternative solution, which does not require writing backend code or a complex setup, is to use a software-as-a-service solution for cloud-based datastore. These services normally provide JavaScript API access and support real-time synchronization through event callbacks or through plug-ins for specific client-side frameworks. Using the JavaScript API directly, the developer would need to write a significant amount of code to ensure that the interface is kept in sync with the datastore changes and user interaction. Some of the boilerplate code that would be required is:

- initialization of the connection to the datastore service
- event callback for adding new task that sends the data to the datastore
- event callback for updating existing task in the datastore
- event callback for changes coming from the database that updates the task list and at the same time creates the previously mentioned update event callback

Figure 1 shows the code needed for implementing ToDo App with Endev. It does not require any JavaScript code to be written by the developer. It still uses a datastore service, but now the JavaScript API access to the datastore and its event callbacks are hidden behind declarative annotations (e.g. data-from, data-where, etc.). These annotations contain only the necessary information: what is the data in question (i.e. TodoList), what is it bound to (i.e. input tags) and what does the user interaction do (i.e. inserts a new task to the TodoList). Furthermore, the same annotations can be used for accessing the data from web services as will be demonstrated by our second example application.

#### B. Photo Collector App

Figure 2 shows the interface and the code of the Photo Collector application that allows the user to search for Flickr images and add them to a personal collection by clicking on an image. The application uses two Flickr APIs, one for finding photos related to the search term and the other, called once for each photo, to get the username of the photo’s owner.

Without Endev, the first challenge in developing the Photo Collector would be accessing the API information from the client-side code since Flickr APIs do not support the Cross
Origin Resource Sharing (CORS). One option to overcome the issue would be using backend code and a web server to access the required API, however, that is not an option for rapid prototyping or end-user development. Instead, the solution could be to use the Yahoo Query Language (YQL) platform which offers a common interface to a multitude of web services. YQL executes the API calls in the backend and returns the data through its CORS enabled REST interface.

By using Endev we can again completely remove the need for JavaScript code and use only annotations to access the web service data. Given the YQL capabilities to avoid CORS and to combine multiple web services, which are both needed to enable prototyping, we have decided to make YQL one of the Endev data providers. The bottom of Figure 2 shows the Endev annotations of the Photo Collector application.

The hierarchical organization of the annotations reflects the dependency of the API calls and removes the need for writing error-prone nested callbacks. In other words, only when `flickr.photos.search` data is retrieved will the `flickr.people.info2` annotations be triggered.

Finally, the developer needs to implement a solution for storing a selected image, which has the same challenges as the ToDo application with regards to complexity of storing users’ own data. However, using Endev we can easily use the retrieved data from the web services and previously seen datastore functionalities. Adding `data-insert-into` and `data-click` annotations, we have finished the task. The last four lines in the code in Figure 2 list all the pictures in the collection.

### IV. Key features and implementation

In this section we present the key features of Endev that enable prototyping real-time collaborative web application and web service data mashups without the need of writing JavaScript code or setting up a backend.

#### A. Client-side only, no backend required

Endev aims to support rapid prototyping and end-user development of web applications populated with real data. Thus, one of the goals was to remove the need for a backend. Therefore, the only requirement for starting Endev application development is to include the Endev library code in the HTML file. No other boilerplate or bootstrapping JavaScript code is required, instead the developers start by writing HTML annotations.

To reduce the threshold for prototyping, Endev employs cloud-based services. Firebase is used as a datastore service and YQL as the CORS enabled interface to any web service data. Endev abstracts the API interaction with the datastore and YQL into a single set of SQL-like data-binding HTML annotations, thus removing both the need for setting up a dedicated backend and writing JavaScript code to access cloud-based services.

An additional benefit of having a fully client-side application is that developers, designers and end-users can easily collaborate by exchanging the HTML file (and if needed other libraries and resources) by any file sharing method they prefer (e.g. via email, Dropbox, etc.). Furthermore, the code can easily be put on one of the free code playgrounds (e.g. CodePen, JsFiddle) which can then be used to collaborate on development or simply used for sharing the access to the app.

#### B. On-the-fly data creation, no data model required

On-the-fly data creation allows Endev users to define new data entities and/or their attributes as they need them without modifying the database schema or abstract model representation. A button that, when clicked, adds a new task into a to-do list can be realized with the following HTML code annotated with the Endev attributes:

```html
<button data-insert-into="firebase:TodoList" data-click="insert({name:'Write a paper', done:false })">Add task</button>
```

When the user clicks the button, Endev will automatically create a TodoList collection, if one does not already exist, in the Firebase datastore and add an object with name and flag indicating task completion. This is achieved by using a document-based NoSQL datastore that supports semi-structured data (e.g. JSON-like objects) without the need of having predefined database schemes.

#### C. Direct three-way data-binding, no controller required

Data-binding constraints that are declared only once and then maintained automatically have been a matter of research for some time [21], [29], and adopted in popular web development frameworks [2], [3], [7]. With state of the art data-binding solutions like Angular, developers need to define the model and the data connection in JavaScript, which can be problematic for non-skilled developers or can lead to losing focus during prototyping. Furthermore, additional code is needed to bind a specific model to a specific view (e.g. through controllers). Finally, when working with the layout, the developers need to remember what the model exactly represents (i.e. what is the real source of the data).

To address these limitations for prototyping and non-skilled developers, Endev introduces the concept of **direct data-binding**. Instead of binding the DOM elements to abstract model representations, the developers directly write the data source (e.g. SQL-like query over datastore entity or REST endpoint) in the data-binding HTML annotations. To continue on the to-do list example, if we would want to list all the items in the TodoList collection which are not yet completed, we could write:

```html
<div data-from="firebase:TodoList item" data-where="item.done = false">
  <b>{{item.name}}</b>
</div>
```

Here, Endev uses the `data-from` annotation to define: the source of the data (`TodoList`) and a label (`item`) that will be used to represent the returned result. A `data-where` annotation is used to filter the results. The HTML element containing the `data-from` annotation will be duplicated for each item returned from the collection. The double curly brackets represent Angular templating that serves as one-way data binding i.e., displaying the value of the task’s name.

Endev leverages the real-time support of the datastore to keep the data-binding in sync with the information from the datastore, meaning that if another instance of the interface
would be running on a different browser and a new item would be added or an existing item would be changed, this list would be updated automatically.

If we want to expand the example, to be able to edit the items in the list, we just need to replace one-way data-binding with appropriately annotated input boxes and an update button:

```html
<div data-from="firebase:TodoList item"
    data-where="item.done = false">
  <input data-value="item.done" type='checkbox'/>
  <button data-click="update(item)" data-value="Save"/>
</div>
```

We could also decide to forgo the save button and instead have the changes be automatically propagated to the datastore as they happen in the interface i.e., the three-way data binding. To enable three-way data binding we add the annotation data-auto-update:

```html
<div data-from="firebase:TodoList item"
    data-where="item.dueDate < Date()"
    data-auto-update="true">
  <input data-value="item.done" type='checkbox'/>
  <input data-value="item.name"/>
</div>
```

Direct data binding can also be applied to web service data, in such cases currently only one-way binding is supported. In other words, the web service data is read-only. Examples of web service bindings will be shown in the section that follows.

D. Hierarchical and multi-domain queries

Web services often return JSON or XML, in other words, objects that can have nested objects and collections. In the Photo Collector example (see Figure 2) the `flickr.photos.search` API returns an object that contains an array, called photo, of objects representing the Flickr images. To be able to display each object (image) from the array we need another, nested, data-from annotation:

```html
<div data-from="yql:flickr.photos.search result"
    data-where="result.photo.photo"/>
</div>
```

If there is a need to access more than one web service and their calls are dependent, i.e., the results of one web service call is used as an input for the other, the needed JavaScript code would normally have unwanted nested callbacks. Such interdependent web service access can be achieved with Endev by specifying the condition of the dependency between the data in the data-where annotation.

Finally, the data combination and nesting can be achieved independently of which service or datastore the data comes from. This is demonstrated with the Photo Collector example where data retrieved from the Flicker service with the YQL provider is used in the insert statement with the Firebase provider.

E. Helper annotation

When working with data coming from APIs, one of the challenges is to understand how the data is structured. Endev offers an extra annotation data-explain that helps developers understand the data they are working with. Taking as example the call to Flickr API from Photo Collector app, the developer could add the following annotation:

```html
<//..omitted..>
```

Endev would then, instead of displaying the defined HTML content inside that element, display nicely formatted raw data that the API returns.

F. Implementation

Endev is implemented as JavaScript framework and its architecture can be seen in Figure 3. Endev Core is responsible for sending the Create-Read-Update-Delete (CRUD) operations to data providers and for keeping the client-side in-memory representation of the received data (data cache).

Endev Annotations, two-way data binding and interaction events (e.g. data-click) are implemented on top of the Angular [7] framework. The two-way data binding is established between the HTML annotation and the in-memory data representation (data cache) in Endev Core. Furthermore, Endev Core uses the annotation information to compose the queries needed for data access (including resolving the dependencies in case of nested queries). For each query a correct provider is identified (e.g. Firebase, YQL) and invoked.

Endev Providers are responsible for accessing each individual data source supported by Endev. They implement the data source specific CRUD operations (or just Read in some cases). If the data source supports sending events when data changes on the source (i.e. real-time synchronization) the Provider will subscribe to such events and trigger the data cache update in Endev Core. Currently there are 4 Endev Providers implemented, but more can be added by 3rd parties:

- **Local Storage** is a wrapper over the HTML5 localStorage feature that supports storing information in the browser. The data is stored only on the client, therefore it can be a good candidate for quickly prototyping or local development, but not for data that needs to be shared or accessed from different browsers.

- **Firebase** is a provider for accessing the cloud data store. It utilizes the API of Firebase that also supports real-time synchronization.

https://www.w3.org/TR/webstorage/
YQL accesses the YQL service that has built-in support for various APIs. Furthermore, YQL can be used as a proxy for accessing any other JSON or XML API. Currently only Read operations are possible with YQL provider.

**REST** provider is used to directly access any JSON and XML API that supports CORS. This can be used with APIs that for some reason do not work with YQL (e.g. have a response time bigger than 30 sec). Currently only Read operations are possible with REST provider.

**G. Extensibility and beyond annotations**

Since Endev annotations are implemented in Angular, all the data retrieved by Endev can be used in regular Angular expressions and third-party plugins. This empowers the developer to use advanced layout components like calendars, spinners, etc. by just passing Endev-retrieved values to their HTML attributes. Furthermore, it opens up the possibility of building evolutionary prototypes as developers can use Angular or pure JavaScript code for features beyond Endev’s capabilities and, if needed, gradually replace Endev annotations altogether. Such tasks would require additional learning from unskilled developers, but such learning would be only incremental (e.g. new HTML tags) as opposed to starting from scratch in completely new technology.

Endev Providers are designed to be modular and thus new providers could be added at any time. This could be used to, for instance, support a dedicated backend, Google Spreadsheets, or open source PouchDB [6] as real-time datastore instead of Firebase.

Finally, there is often a need for more complex data formatting and using libraries that work outside a specific framework (i.e. Angular). For this reason Endev-retrieved data can be passed through user defined JavaScript functions where it can be processed and used as desired.

**V. Evaluation**

We conducted an evaluation of Endev with the aim to investigate whether people with experience in building interactive systems would perceive Endev as a useful and easy-to-use tool for prototyping and in which situations.

**A. Participants & Method**

The study involved 15 participants. The demographic data showing age, gender and previous years of programming experience can be seen in Figure 4. We recruited participants from our professional network where the invitation to participate was shared either directly over email or through social media. We looked for individuals who through their work or hobby develop web applications or are, in other ways, involved in designing interactive systems. All participants had some programming experience and experience writing or modifying HTML. None of them had used Endev before.

Each person participated in the study individually on-line and in all except one case they completed the study without direct observation. The study setup recorded their activity including the start time and their progress through the study.

The central part of the study was an interactive Endev tutorial\(^3\) that guided participants through the main features of Endev. The tutorial presented code examples using on-line code playground CodePen\(^6\) where participants could see both the code and the resulting interactive interface. Furthermore, by opening the example in the new window they were able to modify the example and see the resulting interface side-by-side. During the tutorial they were encouraged to modify the example code in order to test certain options and features.

There were 5 tasks throughout the tutorial reflecting the features covered in each part and growing in complexity as more advanced features were introduced. The tasks can be categorized as:

- tasks 1 and 4 required re-applying existing knowledge already covered in the tutorial
- task 2 required combining existing knowledge in a new way
- tasks 3 and 5 required discovering new features and applying existing knowledge in an advanced manner

The tasks were optional as our main aim was to gather data and feedback through participants’ comments and the survey. The aim of the tasks was to additionally evaluate if Endev concepts are understandable in the form presented throughout the tutorial.

Any modification made to the code by participants was automatically recorded. This included both modification to the tutorial examples, which they were guided to do, and all of their attempts at solving the tasks. Finally, they had options to provide extra comments for each task. The interface for providing comments was implemented with Endev.

At the end of the tutorial participants were asked to complete a survey using Google Forms. The first part of the survey contained demographic questions and questions assessing their background experience including self-assessment of their programming knowledge in relevant technologies (i.e. HTML, CSS, JavaScript, SQL, NoSQL and front-end libraries) according to Bloom’s Taxonomy [13].

The second part of survey consisted of Davis’ Technology Acceptance Model (TAM) [18] questions to assess how participants perceive Endev with respect to Usefulness and Ease of Use. According to Davis, subjective measures of Ease of Use and Usefulness are more relevant to users’ final decision if to use the technology than objective usability measures such as task completion time and error rates.

The last part of the survey contained open questions asking participants to compare Endev to other technologies they have

---

\(^3\)http://www.endevjs.org/tutorial-v01

\(^6\)http://codepen.io

---
used, describe scenarios and for what type of applications they would use Endev and, finally, give any other comments they might have.

B. Results

All of the recruited participants went through the whole tutorial and finished the survey. Out of the 15 participants, 10 (referred to in the following text as P1 to P10) also attempted to finish the optional tasks. The average time to finish the tutorial (not counting the survey) was 76 minutes in total, or 82 minutes for the ones doing the tasks and 57 for the ones not attempting to do the tasks.

In order to get enough confidence that the TAM survey questions were answered with at least a minimum understanding of how Endev works, we have analyzed the tutorial task completion progress. We included in this analysis the recordings of their code sessions to understand to what extent a task was completed or why the approach to complete it was wrong.

The fact that 5 participants did not attempt to solve any task is not worrisome because they were among the more experienced programmers (4 of them had more than 10 years of experience and the 5th had more than 6) and combined with their qualitative comments we believe that they gained enough understanding to be able to answer the TAM questions without needing to actually write the code.

The overall task progress of the other 10 participants is shown in Figure 5. We can not regard not-started (NS) tasks as an indication of task difficulty because participants reported in their comments that the tutorial took too long and they preferred to move to other parts of it instead of attempting these tasks. We have defined partially completed (PC) tasks as tasks where our code analysis indicated that only a minor improvement would have brought the correct solution, while the core part of the task was completed correctly.

The results suggest that participants interacted with Endev to a sufficient extent to give us confidence about their TAM answers (in the case of one participant (P5) our code session recording failed during the tutorial so the participant might have engaged with more than 2 tasks). While the number of completed and partially completed tasks is encouraging and the code analysis of all tasks gives us input to our design, we turn to the TAM analysis for Endev assessment.

C. TAM assessment

A TAM analysis involves a corroboration of Usefulness (Figure 6) and Ease of Use (Figure 7). The aggregated mean value score for Usefulness is 5.28 and 5.78 for Ease of Use, both being high values on the 1-7 scale. The Alpha test indicates strong internal reliability for our Usefulness data (0.93) and acceptable internal reliability for the Ease of Use (0.74). As the alpha test takes the sample size into account, this result suggests that our small sample size did not affect the reliability of our results. We will now comment on the Usefulness and Ease of Use data that we collected.

1) Usefulness: Figure 6 shows the diverging stacked bar plot of the answers to Likert-scale TAM questions focused on perceived Usefulness.

As the batch of Likert-scale questions were complemented with open ones, we have performed a category analysis of the qualitative answers. Participants generally expressed that Endev is most useful when there is a need to do something quickly. Speed of creating the interface is expressed in different ways: rapid (P2), quick (P3, P11), “whip up a relatively simple website, but do it quickly” (P8), “little time and effort” (P10). In general, this suggests that Endev gave users a sense of fluid user interface creation which is in line with our design intentions.

As for characterizing the domain in which Endev is useful, P1, P2, and P14 explicitly mention prototyping. Many respondents explicitly emphasize prototyping for themselves (P4, P10, P11) and others imply own use indirectly like “testing a [data] API that I write myself, to see if it is usable” (P2). Playful design exploration is best expressed by P2: “Have fun, poke around a few data services”. P4, who is generally skeptical to prototyping as he regards himself experienced enough with more mainstream, production-level libraries, still sees a niche for experimental use “if I would be building a simple demo for myself, e.g. an interface for a raspberry pi weather control... or a dashboard to aggregate different metrics from different sources or so”.

Besides prototyping for own use and reflection, our informants suggest that prototyping could be done for clients (P2), students (P11) or participants in customer requirements analysis (P11). In regard to who else could be the prototyper, P14 suggests “marketing/designer types” and also beginners.

Rationales for why Endev is useful for prototyping are also given. The setup-free aspect of Endev is emphasized as “without the need for e.g. setting up SQL servers etc.”
“Ender would help me trim the grunt work.” (P8). P14 emphasizes the JavaScript-free aspect, i.e. making an interactive prototype richly populated with web and own data without the need to write procedural code: “it’s nice to have a way to “play” with html and see it “do things” without having to write code - this is important for beginners.”.

Limitations are also emphasized by informants, starting from prototyping itself. One experienced engineer (P12) does not believe in interactive prototyping and in throwing away prototypes, which relates to developer skepticism to prototypes observed and regarded as unjustified by Beaudoin-Lafon and Mackay [10]. P1 raises security issues, expressing that Ender is useful “where security and confidentiality are non-critical since I have little control over users’ data (stored in the cloud)”. Furthermore, other participants (P4, P11, P12) were also interested to see how Endev would scale to bigger applications or how one would go from Endev prototype to non-Endev application.

From these results we can conclude that overall Endev is deemed useful for prototyping applications that are small or with limited scope (e.g. only presenting data, or for personal use by the “developer”). In such situations even more skilled developers see the benefit in using Endev as it allows the developer to be quick. The challenge for Endev with regard to prototyping larger scale solutions is the path from the prototype to the fully functioning application. Answering these questions is beyond the scope of this paper, however, some possible directions to addressing this are presented in future work.

2) Ease of Use: The diverging stacked bar plot for Ease of Use is shown in Figure 7. While the Ease of Use mean (5.78) is higher than the Usefulness mean, the internal reliability is just above the acceptance threshold, which is most probably due to the lower scores related to flexibility. We are still deliberating how to interpret this result.

When comparing to other libraries that they used, informants characterize Endev’s Ease of Use by emphasizing the lack of need for JavaScript (P2), for “boilerplate” (P2) and for “data manipulation logic” (P5) or for controllers (P2, P15). P2 further emphasizes that needing such code would get the developer “side-tracked”, hence Endev allows one to “focus (...) on what matters”. Informants also mention the practical, low-setup aspects “directly embedded in the DOM (...) I can do everything in the browser and in one single HTML file” (P2), the easy access to open API data “access is easier than e.g. angular” (P5) and “Out of the box real time support is also cool” (P15).

Other respondents emphasize learnability aspects such as “easier to understand [than other frameworks]” (P11). A novice developer says “My experience is somewhat limited, but in 1h of trying Endev out, I managed to do more things (and in a simpler way) than in other libraries.” (P8). This is confirmed by a person who has left the software development area 7 years ago: “easy [...] and pleasant and generally makes me want to give it a shot. My main issue at the moment is lack of time to pursue some of my ideas for personal projects, but with Endev I would be more inclined to start working.” (P10). Such remarks encourage us to believe that we have created a low threshold in this domain, which is important for prototyping.

Experienced developers feel that they are rapid enough with certain production technologies to prototype directly using those. P4 says “if I needed to make a quick prototype, I would use the tools I know to build it. Based on previous experience [with learning new frameworks], I’m afraid with Endev I would just spend a lot of time ironing out some weird corner case bugs that I can’t foresee right now.”. Indeed, it is recommended [10] to prototype with the material that lets a developer think more of the prototyping problem than on the prototyping tool, and Endev specifics may constitute blockers in that smooth process.

VI. CURRENT LIMITATIONS AND FUTURE WORK

Several participants in our study expressed concern about how would one go from an Endev prototype to a non-Endev application. While we believe there would be no need to replace Endev even for more complex applications, this has not yet been extensively explored beyond early promising results.

With our goal to support prototyping and end-user development, we were not concerned with potential security threats, performance and optimization issues and consider engineering these issues as a separate task. In our previous work [14] we have managed to address these concerns and are confident we can do so with Endev as well.

Finally, our evaluation with TAM was done on a small sample size that gave us good indication and valuable input for the future work. To confirm our results we are planning to conduct a evaluation with a larger sample size and with less skilled developers. To achieve that we can reuse the Endev tutorial, improved with the lessons learned from this study, that can easily scale to a larger number of participants.

VII. CONCLUSION

This paper has introduced the Endev framework, which provides a novel approach to rapid web application prototyping that need to access data from various sources without the need to write JavaScript or other procedural code. It introduces declarative SQL-like annotations applied to GUI elements and serve as direct data-binding to data source entities - web services or the application’s own data. Furthermore, it utilizes cloud-based services to remove the need for having a backend. In other words, Endev-annotated HTML files are fully functional applications that can be easily shared among collaborators. Finally, a TAM study has shown that developers perceive Endev as a useful and easy-to-use tool for prototyping.
REFERENCES


