Universität Osnabrück
Fachbereich Humanwissenschaften
Institute of Cognitive Science

Bachelor thesis

Designing a modular architecture for building web-based psychological experiments

Marcel Klehr
977686
Bachelor's Program Cognitive Science

First supervisor: Prof. Dr. Michael Franke
Institute of Cognitive Science
Osnabrück

Second supervisor: Britta Grusdt, M.Sc.
Institute of Cognitive Science
Osnabrück

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

city, date

signature

In the social and behavioral sciences, web-based experiments allow experimenters to recruit larger, more diverse subject pools with higher statistical power, and can speed up experiments dramatically, while being portable to any device with a browser. While web-based experiments have become increasingly popular, methods to construct them have been lacking in flexibility and simplicity. We introduce magpie, a modern, component-based, declarative JavaScript framework based on Vue.js for building web-based experiments of varying complexity with ease, including the possibility to have participants interact virtually.

Contents

1 Introduction

Web-based experiments are gaining more popularity in behavioural research because of their flexibility and efficiency. I conducted an analsysis of prior art and found that many existing tools for realizing web-based experiments lack sufficient flexibility and expressiveness. This lead to the creation of magpie, both a server for managing experiments and a client-side framework for implementing them.

Magpie experiments can be run directly on a local computer and published online with ease. They can also be shared in an editable format and reused or adapted, enabling transparent replications and facilitating open science. Magpie is provided free of charge under an open-source license.

In this work, I initially examine prior work and competing software products with a set of requirements in mind. In the following section I explore the design goals and choices in relation to these requirements that lead to the implementation of magpie's client-side framework, its concepts, components, features and technical realization. Afterwards, I discuss limitations of the current design and directions for furture work. Finally, I reach conclusions about the design and its applications.

In addition to this document, this thesis includes the creation of the full online documentation of magpie available at `https://magpie-manual.netlify.app` and `https://magpie-reference.netlify.app`.

1.1 Web-based experiments

Psychology and Social sciences have traditionally conducted research in close proximity to participants: in laboratory settings, after the fact in natural experiments or directly in the field. These modi have their limitations of capacity, speed, and subject diversity,

especially when targeting a specific, possibly rare or remote population. The recent COVID-19 pandemic also severely inhibited direct-contact experiments due to contact-restricting preventative anti-infection measures.

Web-based experiments, by contrast, allow participants to take part in an experiment via the internet. This generally allows experimenters to recruit larger, more diverse subject pools, providing higher statistical power, and can speed up experiments dramatically due to their unsupervised nature.

As participants are usually unmonitored, however, experimental instructions may be ignored or read too carelessly, potentially leading to lower quality data. Researchers are additionally largely unable to control the environment, potentially incurring significant distractions during the experiment. Technical details such as network connection speed and reliability, computer and browser types as well as screen resolution may contribute to additional variance in the data.

Although these uncontrollable factors suggest decreased reliability of web-based experiments, experiments on LabintheWild have been shown to replicate previous in-lab results. (Gajos et al., 2015) In similar fashion, Schoeffler et al. (2013) compared results from laboratory and web-based studies and found no significant differences. Ryan et al. (2013) even concluded that unsupervised web-based experiments with large subject pools have benefits over smaller, supervised lab experiments that outweigh their potential costs with their larger sample size. Specifically, patterns that were not significant in the lab were significant on the web.

2 Requirements

In our analysis of prior work as well as in the design of magpie, we focus on the following features.

An **Open Source** license is essential for common access to the tool in question as well as sustaining development. **Recruiting flexibility** requires that the tool in question does not restrict recruiting to a specific participant recruiting platform (e.g. Amazon Mechanical Turk or Prolific). Tools should offer a **Framework** for building custom experiments rather than a mere library to select experiments or trials from. **Implementation flexibility** refers to a Turing complete programming environment. **Shareable customizations** are important to facilitate re-use of custom experiment parts across experiments. **Interactivity** indicates the ability of a tool to allow participants to interact online during an experiment, both synchronously and asynchronously. **Dynamicity** indicates the ability to use a participant's past responses to control the course of the experiment. This is useful for applying Optimal Experiment Design that takes into account participant's responses in a variation of Ouyang et al. (2016). **Multimedia support** indicates the possibility of including image, audio and video stimuli in experiments. A **Catalogue** of a wide variety of building blocks necessary for different experiment and task types.

3 Analysis of prior work

There are a variety of pre-existing software packages attempting to aid in the construction of web-based experiments. In the following we examine some prominent candidates.

OpenSesame (Mathôt et al., 2012) was mainly designed to be used for offline lab experiments in the fields of psychology, neuroscience and experimental economics. It is licensed under the open-source GNU General Public License and runs across various platforms utilizing Python.

Using an extension called OSWeb, experiments built with OpenSesame can be run in the browser, albeit with somewhat limited functionality. The fact that OpenSesame was designed to be run with Python naturally limits the range of features provided in the browser as not all functionality is ported to the browser. The main strength of OpenSesame, its graphical editor, is also its main weakness: Realizing more complex experiments is often non-trivial with a certain amount of programming knowledge still necessary (e.g. for validating form input, counterbalancing, custom layouts, programmatically generating loop tables). While dynamic experiments are possible via code blocks, it doesn't offer interactivity between participants.

PsyToolkit (Stoet, 2012) is an all-in-one solution for web-based experiments, offering study design, hosting and even data analysis. It appears mainly geared toward students who need a simple solution to run their experiments and implements a custom imperative, rather simplistic scripting language for realizing experiments, which doesn't allow for much experimental complexity or customization. It does not allow for interactivity or dynamicity.

psiTurk (Gureckis et al., 2016) is mainly a server-side framework for integrating web-based experiments with Amazon Mechanical Turk and cannot be used with other recruiting platforms. It only aids minimally in developing the actual experiments and thus

has to be combined with a different front-end framwork.

jsPsych (de Leeuw, 2015) in contrast offers a JavaScript interface to build experiments, theoretically allowing implementers to take advantage of the power of the web platform. It interoperates with JATOS (Lange et al., 2015) as a backend and offers a wide range of pre-built templates called "plugins" to be used for experiments. However, beyond the existing templates, implementing custom functionality and recombining different templates is non-trivial as there are no built-in components and no simple built-in framework for assembling custom trials. Interactivity and dynamicity are not offered.

LIONESS-lab (Giamattei et al., 2020) is a full-featured solution for developing interactive experiments, offering a graphical experiment builder and automatic testing of experiments. While experiments can be equipped with more complex logic by virtue of JavaScript code, the project's scope and built-in building blocks focus largely on economic experiments and do not appear to be extensible. Runnable experiments are a combination of server-side management code and client-side code which requires technical setup effort and skills for each new experiment, making self-hosted deployment non-trivial. However, a free hosting service is available. LIONESS-lab also offers interactivity and dynamicity.

Similar to LIONESS-Lab, **Labvanced** (Scicovery, 2018) offers a full-featured graphical experiment editor, albeit with a much wider range of building blocks for various stimuli as well as interactivity. Labvanced is proprietary and relies on a subscription model.

z-Tree (Fischbacher, 2010) is a popular framework for conducting economic experiments, written in C++ and is thus not web-based although it can be made to work via the internet. z-Tree allows for rather complex experimental setups, but its scope and components are focused on economic experiments.

Dallinger (Suchow, 2016) is a server side experiment manager for automatically recruiting participants from Amazon Mechanical Turk, allowing to group them and facilitating information exchange in interactive experiments. Dallinger has a minimalistic client-side JavaScript API, requiring the use of third-party libraries for implementing rich web experiments. Experiments are modeled with a complex server-side framework that allows automating participant interactions for testing purposes, but requires sophisticated software engineering efforts to build experiments.

	OpenSesame	PsyToolkit	psiTurk	jsPsych	LIONESS-lab	Labvanced	zTree	Dallinger	lab.js	magpie
Web	✓	✓	✓	✓	✓	✓		✓	✓	✓
Free software	✓	✓	✓	✓	✓		✓	✓	✓	✓
Reruiting flexibility	✓	✓		✓	✓	✓				✓
Sharable customizations				✓		?	?		✓	✓
Framework	✓	✓			✓	✓	✓		✓	✓
Catalogue	✓	✓		✓	✓	✓	✓			✓
Interactivity					✓	✓	✓	✓		✓
Dynamicity	✓				✓	?	?	?	?	✓
Multimedia	✓	✓		✓	✓	✓	?			✓
Implementation flexibility	✓	✓		✓	✓	✓			✓	✓

Table 3.1: Overview of features

lab.js (Henninger et al., 2020) is a client-side framework featuring both an application programming interface and a graphical user interface for building web-based experiments. It interoperates with JATOS (Lange et al., 2015) as a backend. Although the graphical experiment builder simplifies usage greatly, lab.js only offers few basic built-in components and doesn't build upon existing JavaScript frameworks for simplicity and efficiency. It does not offer interactivity.

See Table 3.1 for an overview and comparison with magpie. Question marks indicate features that could not be determined from publicly available information. From my analysis, I find that many existing tools for realizing such experiments lack sufficient flexibility and expressiveness, providing only a limited set of components or commands that can be used, often without the ability to recombine them. Some do provide the ability to build custom components, but only with a steep learning curve. Others offer a graphical editor to gain simplicity, while sacrificing flexibility.

4 Design

4.1 Overview

Magpie consists of two parts that each have their own responsibilities: a server-side service and a client-side framework. These two parts, although integrating with each other, are largely independent. The server handles experiment management, configuration and data storage. It is designed to be the backend for multiple different experiments that are possibly hosted elsewhere.

The client-side framework builds upon Vue.js (You, 2014), a reactive, declarative and reactive component framework for the web, to allow the assembly of experiments as single-page web applications that connect to the backend server to store results and exchange information with other participants for interactive experiments. The client-side framework acts both as a component library providing building blocks for experiments as well as an API to the backend server and other utility functions.

From my analysis, I find that many existing tools for realizing web-based experiments lack sufficient flexibility and expressiveness. In the spectrum between usability and flexibility, I believe magpie's client-side to be a worthwhile compromise: Utilizing the expressiveness and simplicity of Vue.js, magpie provides a large catalogue of components that can be recombined and customized to build experiments of any complexity. This makes simple experiments barely harder to implement than writing normal HTML, while scaling to the complexity of, for example, a fully interactive behavioral economics experiment.

4.2 Design goals and choices

A main goal of designing magpie was to provide implementers with a conceptually simple framework for building experiments that should be employable as a tool for teaching the basics of experimental design. At the same time the framework should be generic and expressive enough to facilitate building complex interactive experiments without major software engineering skills.

A key concept for bridging this divide is modularity and composability: Offering a catalogue of building blocks in varying granularity to ease assembly of simple experiments with larger, more rigid components, but also allowing intricate customization via smaller, flexible components that integrate with each other.

To this end, taking advantage of Vue.js allows for fast progress and interoperability with a large catalogue of open source modules already built on that same framework, while avoiding mistakes that others have already made and corrected. Choosing a reactive framework is a major benefit for implementation effiency, because the view layer updates automatically based on the provided data, reducing the amount of necessary code.

A graphical editor is explicitly not part of the scope of this project at the moment. The emphasis on free customizability necessitates the use of a programming language rather than the limited freedom of a graphical editor. However, building a graphical editor on top of magpie is still possible in future work.

Another goal was easy interoperability with participant recruiting platforms such as Amazon Mechanical Turk and Prolific.

4.2.1 Conceptual Framework

A magpie experiment (see Figure 4.1) firstly consists of the data used to build the experiment, also known as independent variables. This could be textual, visual or auditory stimuli that are presented to the participants or possible choices that the participants can make.

Every experiment is further composed of a series of screens that participants go through sequentially. Such a screen could display instructions on how to participate, or present an actual trial task to the participant. Usually one trial will be implemented per screen.

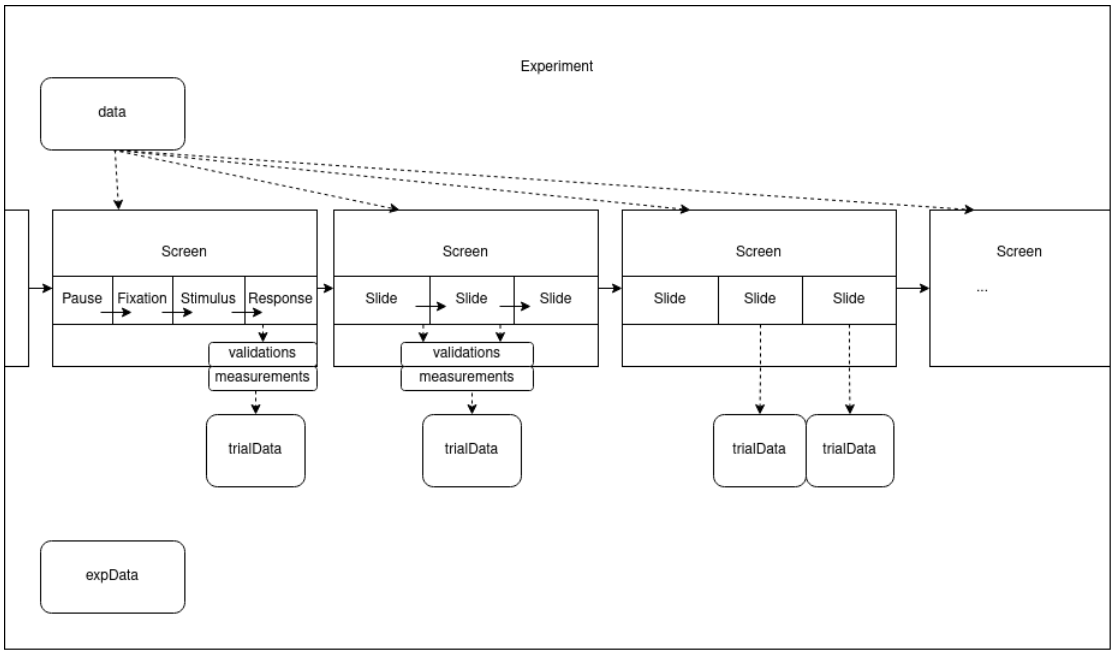

Figure 4.1: Conceptual schema of a magpie experiment

This is similar to what researchers are used to from OpenSesame (Mathôt et al., 2012) and a natural representation of many experiment paradigms.

A screen consists of a sequence of one or more slides. Slides simplify dynamic presentation of more intricate tasks. They are also displayed sequentially, giving experimenters the opportunity to display fixation points, or enforce pauses between stimulus and response. This design was inspired by computer-based psychological studies such as Todd & Marois (2004). A slide can contain arbitrary components, from stimulus-presentation, over an interactive chat, to recording responses e.g. with a rating scale.

Anytime during a screen, result data rows can be saved directly which will be sent to the magpie server when the experiment is completed. To make collecting measurements across slides easier, data can also be gradually accumulated in a per-screen ‘measurements‘ object. This allows validating the data and saving all data in one go, once a trial is completed.

4.3 Components

Magpie provides basic stimulus components like playing audio files, video files, displaying image files, text, a self-paced reading component, a tone Synthesizer and a canvas for

displaying dynamic visual stimuli. Responses can be recorded using dropdowns, a forced choice input, a single and a matrix multiple choice input, a slider, a rating scale, a free text box, a key press input, a constant sum range slider, an image selection input, a text completion input and a rank order input.

These finer grained components offer essential features that can be recombined in a variety of ways to form trials of any complexity. To ease the building of experiments, there are many pre-built screen components available. These allow the quick assembly of simple experiments out of coarse building blocks, which utilize the above fine graned building blocks under the hood. These trial screens follow a life cycle of five slides: pause, fixation, stimulus presentation, response collection and feedback. All phases except response collection are optional, and all phases can be configured with a timeout or a manual trigger to go to the next phase. Additionally, implementers can add textual context like a Question. There are also slots available for each of the four life cycle phases, that can be filled with custom content.

All screen components allow setting a screen title and a progress percentage which will be displayed in a progress bar.

In addition to normal response inputs, magpie also features eye and mouse tracking in order to measure the participants' thought process in more detail.

4.4 Dynamic experiments

Iterated experiments allow using data from past participants in later trials for new participants. This allows the creation of, for example, iterated narration experiments (similar to the game Telephone). Magpie requires three basic settings on the server for running an iterated experiment: The number of variants, which defines how many different experimental settings there will be; the number of chains, which defines how many closed participant groups there will be; and the number of generations, which defines how many participants are in one chain.

4.5 Interactive experiments

Interactive experiments allow participants to interact with each other in some form, for example in strategic economic games, as is frequently described in Game theory. On the client-side, magpie provides components for structuring interactive experiments, such as a screen to wait until the required number of participants in the group are online. To allow the flexible realization of any kind of interactivity, magpie allows sending and listening to events between individual participant sessions. Additionally, it keeps track of which participants currently see the same screen and assigns pseudonymous names and colors to all participants that can be used in experiments for easier identification during interaction.

4.6 Technical implementation

Technologically, the client-side framework is based on Vue.js. It facilitates declarative, reactive rendering and component composition, which is in contrast to traditional imperative programming, and greatly eases reasoning about and implementation of applications. Declarative rendering is facilitated by allowing to extend normal HTML code with behavior-invoking directives and stateful custom elements.

Magpie experiments are compiled using Vue.js's standard tooling, which provides an auto-reloading development server as well as static builds that can be deployed anywhere on the internet.

4.6.1 Example 1: Mental rotation experiment

The following example shows the code for a simple mental rotation experiment. The initial screen would display instructions to the participant, but is excluded here for brevity. Afterwards, the code iterates over a list of trial objects, defined elsewhere, (see line 4) and constructs a new KeypressScreen for each (s. line 5-10). KeypressScreen is a built-in screen with four life cycle phases: Two seconds pause (s. line 8) and one second fixation (s. line 9), as defined in the screen's properties. Third and fourth phase are merged together by default, where the third phase displays the contents of the `#stimulus` slot (s. line 11) and the fourth phase allows responding using the F or the J key on the

keyboard (s. line 10), which will indicate whether the participant thinks the two models in the picture are the same or different. The screen automatically records response time and response. (More information on built-in screens can be found at `https://magpie-manual.netlify.app/01_designing_experiments/00_built-in_screens/`)

After the trial screens are completed the code shows a post-test screen (s. line 18), which asks the participants for demographic information. Finally the last screen will submit the data to the server and thank the participant when everything is completed (s. line 20).

```
1  <Experiment
2    title="Mental rotation task"
3  >
4      <template v-for="(task, i) of trials">
5        <KeypressScreen
6          :key="i"
7          :progress="i / trials.length"
8          :pauseTime="2000"
9          :fixationTime="1000"
10         :keys="{ f: 'different', j: 'same' }">
11         <template #stimulus>
12           <img :src="task.picture" />
13         </template>
14
15       </KeypressScreen>
16     </template>
17
18     <PostTestScreen />
19
20     <SubmitResultsScreen />
21 </Experiment>
```

4.6.2 Example 2: Simon Task

The following example shows the code for a Simon Task experiment. The code in this example leaves out the instructions to the participants. The code iterates over a list of trial objects, defined elsewhere and constructs a new KeypressScreen for each.

The screen will first display a fixation cross for a random duration between 1200ms and 2700ms, specified by the `fixation-time` attribute. Afterwards, the simulus is displayed allowing a response with the Q and P keys. The response is stored in

`$magpie.measurements.response`. The stimuli are displayed using the built-in canvas components, displaying the square or circle left or right of the screen. The Record component adds the trial data to the result measurements. The `response-time` attribute sets a timeout of three seconds to respond, after which the participant is shown the feedback screen which tells them to respond more quickly if they haven't responded yet.

After the trial screens are completed the code shows a post-test screen, which asks the participants for demographic information. Finally the last screen will submit the data to the server and thank the participant when everything is completed.

```
<Experiment title="Simon task">

    <template v-for="(task, i) of trials">
      <KeypressScreen :key="i"
          :keys="{q: 'circle', p: 'square'}"
          :fixation-time="Math.floor(Math.random() * (1500)
             + 1200)"
          :response-time="3000">
        <template #stimulus>
          <CanvasStage
              :config="{width: 800, height: 100}">
            <CanvasLayer>
              <CanvasCircle
                  v-if="task.target_object === 'circle"
                  :config="{
                      fill: 'lightblue',
                      radius: 50,
                      x: task.target_position === 'left' ?
                         50 : 700,
                      y:50
                  }"
              />
              <CanvasRect
                  v-if="task.target_object === 'square'"
                  :config="{
                      fill: 'lightblue',
                      width: 100,
                      height: 100,
                      x: task.target_position === 'left'? 50
                          : 700,
                      y:0
                  }"
              />
            </CanvasLayer>
```

```
32            </CanvasStage>
33            <Record :data="{
34              ...task
35            }" />
36          </template>
37
38          <template #feedback>
39            <p v-if="!$magpie.measurements.response">
40                  Please answer more quickly
41            </p>
42            <Wait v-else :time="0" @done="$magpie.nextScreen()
                " />
43          </Slide>
44        </Screen>
45      </template>
46
47      <PostTestScreen />
48
49      <SubmitResultsScreen />
50  </Experiment>
```

4.6.3 Example 3: Public goods experiment (interactive)

The following example shows the code for a public goods experiment, an interactive paradigm from behavioral economics. The code in this example leaves out the instructions to the participants. Before the trials start, the ConnectInteractiveScreen makes sure the socket connection to the server is active and the necessary number of participants have joined the experiment. The code then iterates ten times, and constructs a new raw Screen for each iteration.

The first slide of the screen will wait for all participants to view the same screen and reset local variables tracking the state of the trial. Once all participants are viewing the same screen, the second slide lets each of them select an amount of their stash to donate to the group using a slider input. When they submit their selection, `submitDonation` is called, which broadcasts the selected value to all participants of this session (s. line 113). The next slide waits for donations from all other participants to have come in, which are recorded by the socket listener in line 88 and checked in line 96 as a computed variable.

Once all participants have donated an amount, the last slide displays the reward they have received as well as their new balance. When the participant clicks on the displayed

button, the results are saved and the next screen will be displayed (s. line 61), which is either another trial, or the post test screen.

After the trial screens are completed the code shows a post-test screen, which asks the participants for demographic information. Finally the last screen will submit the data to the server and thank the participant when everything is completed.

```
<template>
  <Experiment title="Public Goods Experiment">

    <ConnectInteractiveScreen />

    <template v-for="i of 10">
      <Screen :key="i">
        <Slide>
          <WaitForParticipants
            @done="
              resetPool();
              $magpie.nextSlide();
            "
          />
          <p>Waiting for participants...</p>
        </Slide>

        <Slide>
          <p>Your balance is {{ test_stash }} tokens.</p>
          <p>Please select an amount to donate.</p>
          <SliderInput
            :tooltip="true"
            :max="test_stash"
            :response.sync="$magpie.measurements.amount"
            left="0"
            :right="test_stash + ''"
          />
          <template v-if="typeof $magpie.measurements.amount
              !== 'undefined'">
            <p>
              You are donating {{ measurements.amount }}
                 tokens to the group.
            </p>
            <button
              @click="
                submitDonation($magpie.measurements.amount);
                $magpie.nextSlide();
              "
            >
              Go
```

```
            </button>
          </template>
        </Slide>

        <Slide>
          <p>Please wait for all participants to make their
             move.</p>
          <Wait
            v-if="allParticipantsDonated"
            :time="0"
            @done="$magpie.nextSlide()"
          />
        </Slide>

        <Slide>
          <p>
            You have donated {{ $magpie.measurements.amount
               }} tokens and
            received {{ reward }} tokens.
          </p>
          <p>Your new balance is {{ new_stash }}</p>
          <Record :data="{ reward }" />
          <button
            @click="
              $magpie.saveAndNextScreen();
              updateStash();
            "
          >
            Next round
          </button>
        </Slide>
      </Screen>
    </template>

    <PostTestScreen />

    <SubmitResultsScreen />
  </Experiment>
</template>

<script>
export default {
  name: 'App',
  data() {
    return {
      pool: 0,
      donated: 0,
      test_stash: 100,
      amount: 0
```

```
86      };
87    },
88    socket: {
89      donate(amount) {
90        this.amount = amount;
91        this.pool += amount;
92        this.donated++;
93      }
94    },
95    computed: {
96      allParticipantsDonated() {
97        return this.donated === this.$magpie.socket.active.
              length;
98      },
99      reward() {
100       return (this.pool * 1.3) / this.donated;
101     },
102     new_stash() {
103       return this.test_stash - this.amount + this.reward;
104     }
105   },
106   methods: {
107     resetPool() {
108       this.pool = 0;
109       this.donated = 0;
110       this.amount = 0;
111     },
112     submitDonation(amount) {
113       this.$magpie.socket.broadcast('donate', amount);
114     },
115     updateStash() {
116       this.test_stash = this.new_stash;
117     }
118   }
119 };
120 </script>
```

5 Limitations and future directions

A limitation of using Vue.js is the required build step for publishing experiments. Thus, making changes to existing experiments is only possible when the required toolchain is installed. This is a trade-off towards the usability of the live-reloading and modularity features of modern JavaScript frameworks. A key limitation is the need to write code in order to build experiments with magpie, a trade-off we find worthwhile to allow implementers greater flexibility. However, this does not exclude the possibility of building a graphical editor in the future.

Future work on magpie should include measures to improve reliability of experiments, such as "virtual chinrests" (Li et al., 2020), other types of visual display size calibration, the possibility of excluding participants based on the inavailability or inadequacy of certain browser features and recording distractions such as leaving the active browser tab. Additionally, Magpie still lacks built-in screens for a range of trial types, such as random dot kinematograms or visual search tasks.

Another avenue for improvement is interoperability with JATOS (Lange et al., 2015), an experiment management backend, and Sona (Sona Systems Ltd., 2002), a management system for university-assigned experiment participation.

6 Conclusion

Web-based scientific experiments offer higher statistical power at a lower cost and effort with only marginal disadvantages compared to traditional experiment settings. Magpie's client-side leverages the advantages of modern JavaScript frameworks to offer a simple yet powerful framework for building such experiments. Its conceptual structure facilitates realizing basic as well as complex and even interactive experimental paradigms, while striking a balance between usability and expressiveness. Provided under an open source license, magpie facilitates open science, and eases the replication process. Fully relying on the power of a Turing-complete programming language for implementing experiments allows extending the basic framework with additions such as Optimal Experiment Design methods with ease.

Bibliography

de Leeuw, J. R. (2015). jsPsych: A JavaScript library for creating behavioral experiments in a web browser. *Behavior Research Methods*, *47*(1), 1–12.

Fischbacher, U. (2010). z-Tree: Zurich toolbox for ready-made economic experiments. *Experimental Economics*, *10*, 171–178.

Gajos, K., York, K., & Ny, U. (2015). LabintheWild: Conducting Large-Scale Online Experiments With Uncompensated Samples. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, (p. 1364–1378). Association for Computing Machinery.
URL `https://doi.org/10.1145/2675133.2675246`

Giamattei, M., Yahosseini, K. S., Gächter, S., & Molleman, L. (2020). LIONESS Lab: a free web-based platform for conducting interactive experiments online. *Journal of the Economic Science Association*, *6*, 95–111.

Gureckis, T. M., Martin, J., McDonnell, J., Rich, A. S., Markant, D., Coenen, A., Halpern, D., Hamrick, J. B., & Chan, P. (2016). psiTurk: An open-source framework for conducting replicable behavioral experiments online. *Behavioral Research Methods*, *48*(3), 829–842.
URL `https://psiturk.org/`

Henninger, F., Shevchenko, Y., Mertens, U. K., Kieslich, P. J., & Hilbig, B. E. (2020). lab.js: A free, open, online study builder.
URL `https://lab.js.org/`

Lange, K., Kühn, S., & Filevich, E. (2015). Just Another Tool for Online Studies (JATOS): An Easy Solution for Setup and Management of Web Servers Supporting Online Studies. *PLOS ONE*, *10*(7).

Li, Q., Joo, S. J., Yeatman, J. D., & Reinecke, K. (2020). Controlling for Participants' Viewing Distance in Large-Scale, Psychophysical Online Experiments Using a Virtual Chinrest. *Scientific Reports*, *10*(1), 1–11.

Mathôt, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source graphical experiment builder for the social sciences. *Behavior Research Methods*, *44*(2), 314–324.

Ouyang, L., Tessler, M. H., Ly, D., & Goodman, N. (2016). Practical optimal experiment design with probabilistic programs.

Ryan, R. S., Wilde, M., & Crist, S. (2013). Compared to a small, supervised lab experiment, a large, unsupervised web-based experiment on a previously unknown effect has benefits that outweigh its potential costs. *Computers in Human Behavior*, *29*(4), 1295–1301.
URL `https://doi.org/10.1016/j.chb.2013.01.024`

Schoeffler, M., Stöter, F.-R., Bayerlein, H., Edler, B., & Herre, J. (2013). An Experiment about Estimating the Number of Instruments in Polyphonic Music: A Comparison Between Internet and Laboratory Results. In *ISMIR*, (pp. 389–394).

Scicovery (2018). Labvanced.
URL `https://www.labvanced.com/`

Sona Systems Ltd. (2002). Sona Systems Software.
URL `https://www.sona-systems.com`

Stoet, G. (2012). PsyToolkit.
URL `https://www.psytoolkit.org`

Suchow, J. (2016). Dallinger. laboratory automation for the behavioral and social sciences; human culture on a chip.
URL `https://github.com/Dallinger/Dallinger`

Todd, J., & Marois, R. (2004). Capacity limit of visual short-term memory in human posterior parietal cortex. *Nature*, *428*.

You, E. (2014). Vue.js.
URL `https://vuejs.org/`