VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Tutorial Builder
Final Report

Zoltan Debre

Supervisor: Dr Stuart Marshall

October 6, 2016

Submitted in partial fulfilment of the requirements
for
Master of Computer Science.

Abstract

Presenting a computer programming problem or solution with a high-quality video is time-consuming and it is not easy to update.

Technical content producers are struggling to find an efficient and interactive way to show their work.

This research introduces a tool that can be used for creating an interactive tutorial.

Furthermore this tool shows programming code snippets in web based code editor so the content creator can build a playable step by step "movie" with it.

Contents

Chapter 1

Introduction

This project is about building an online publishing tool prototype, using code editors and step by step instructions to present programming challenges and solutions for a computer science problem.

The prototype has two parts, an administration area, where the content creator can build a tutorial, and a player tool, where the recorded steps would be presented.

The primary target user is the creator, who composes a new tutorial. The creator can be a teacher, or an open source project owner, who would like to introduce their tool or code.

The secondary user is the consumer, who wants to learn or know more about a problem or a coding solution.

1.1 Motivation

We all have the unstoppable desire to learn. We are keen to know more about the world around us, about our hobby and our profession. In software development, in computer science, the knowledge is essential, it is the key to succeeding. Reading, studying, sharing. An infinite loop of collecting and adapting new practices.

In information technology, especially in programming languages, writing blog posts, creating static, step by step tutorials are a popular way to share or learn something new. Producing and sharing the content is easier nowadays, but still requires more effort from the creator, when they want to deliver an easy to understand high-quality tutorials.

Creating interactive tutorials are appealing, but the production cost is much higher.

Recording a video tutorial or especially updating it is time-consuming, and it involves more effort from the creator.

I think an ideal solution would be a healthy mix of static and dynamic contents, where the learners can read instructions but meanwhile they can watch the steps in a code editor, in a more realistic environment.

1.2   The problem

When a developer, teacher or hobbyist would like to present a computer programming problem or solution, most of the times they record a video and publish it on YouTube. Sometimes the easiest way to record a video of a conference/meetup presentation or a live coding session. However recording and editing a high-quality video is time-consuming and less flexible. It is also hard to update.

Other problem with showing tutorials with a simple video, that the audience cannot give it a try, they have to configure their computer and environment to play with the presented solution.

Most of the cases we would like to show instructions and code snippets, mainly text-based contents. It is preferred to show code snippets in a more realistic environment, such as in a code editor. Therefore using the online code editor with a pre-scripted way to play the presentation, where the user can navigate back and forth and can modify or play with the code is more interactive. It involves everyone and helps to understand a problem clearly.

Additionally, it is much easier to maintain, upgrade or fix for the content creator.

Furthermore, the user is able to experience it and can see the result, so the new information can be put into practice immediately.

1.3   Personas and their goals

1.3.1   Primary persona

Content creator, open source project maintainer, teacher.

Their motivation is to present a technical problem and the solution in a clear and an easy-to-understand way. The best option is to show a demo, what happens when we insert the suggested code, how easy it is to use. Most of the times a presentation

involves more steps. For example, we would like to show a starting state, maybe a few lines of code which we are able to simplify, so in this case, the first step is to show the problem and after we show step by step how we solve that.

1.3.2 Secondary persona

The consumer, who watch the presentation, who reads the tutorial and who wants to learn more about the actual problem.

They would like to play, stop, and control the presentation. Control the flow, going forward or stepping back.

They would like to try the solution, for example how the final state changes when they modify the code.

1.3.3 Target groups

I develop a prototype web app, where the content producer can create a simple step by step tutorial, and the content consumer can "watch" this tutorial and can interact with it.

There are two different users:

- content producer, for example a teacher
- content consumer, for instance a student

Chapter 2

Related Work

During the development and research work, I found a few related interesting projects. These are similar or I can use them partly.

2.1   CodeMirror Movie

I found this project when I checked the most popular web-based code editor tool, CodeMirror website. The creator of the CodeMirror wanted to present their tool with a realistic way, so CodeMirror Movie was born. [2]

This solution highly coupled with CodeMirror, it is similar to an add-on, so it is possible to attach any CodeMirror implementation. (More about CodeMirror in the next section.)

Adding CodeMirror Movie to our project is straightforward because the open source repository provides a CSS and a JS file, so they can be added to any page.

This tool mainly targets web-developers, so with the help of this tool they can add code and scripts to their websites.

Editing "the movie" script is manual. There is a simple syntax which control the presentation steps and this script should be added to the textarea which will be in the code editor.

We clearly see that it is a very effective way to build a presentation, however it requires real development skills.

Pros:

- simple, lightweight implementation
- easy to add your project if you use CodeMirror and you are a developer
- simple script language to manage the presentation
- user can use the code editor to try the presented solution

Cons:

- mainly for developers only
- highly coupled with CodeMirror

2.2 Comparison of online code editors

There are 3 popular web based code editors: CodeMirror, Ace Editor and Monaco.

CodeMirror and Ace Editor are commonly used on websites and different projects. Monaco is a new solution from Microsoft and it is extracted from their popular Microsoft Visual Studio Code developer tool.

There is not significant differences between them. All has the most important code editor features, like supporting more than 100 languages, autocompletion, syntax highlighting, controlling with shortcuts.

I choose CodeMirror, because it has already Ember.js support. Thanks for the ivy-codemirror Ember addon, it can be added to any Ember.js project with the installation of the addon.

CodeMirror

- Github link: `https://github.com/codemirror/CodeMirror`
- Website: `http://codemirror.net/`
- Popularity (GitHub Star): 9396
- Ember.js Addon: `https://www.emberobserver.com/addons/ivy-codemirror`

Ace Editor

- Github link: `https://github.com/ajaxorg/ace`
- Website: `https://ace.c9.io`
- Popularity (GitHub Star): 12950

- Ember.js Addon: none

Monaco

- Github link: `https://github.com/Microsoft/monaco-editor`
- Website: `https://microsoft.github.io/monaco-editor/`
- Popularity (GitHub Star): 2322
- Ember.js Addon: none

2.3   Reviewing code sharing websites

We can use code editor and sharing platforms also when we want to demo a small feature or describe a problem. These websites are combinations of code editors and an iframe where we can see the preview of the code snippets.

One of the common features is splitting the screen and providing different windows for editing html, css and javascript separately.

User can save the edited content also. Most of them can be embed in a blog post or in other website.

Most important findings:

- All use Code Mirror as code editor
- All of them separate the css, html and javascript editing in different screens, but they merge into one file, and preview of this merged html file is possible in an iframe.
- Saving the different type of code (css, javascript, html) separately.

	Code Pen	JSBin	JSFiddle	Ember Twiddle
Link to open source project	not open source	github.com/ jsbin/jsbin	not open source	github.com/ ember-cli/ ember-twiddle
Website	codepen.io	jsbin.com	jsfiddle.net	ember-twiddle.com
Code Editor	Code Mirror	Code Mirror	Code Mirror	Code Mirror
Embedding support	yes	yes	yes	yes

Chapter 3

Design and Implementation

3.1 The technology of choice

One of the most popular programming languages in web development is JavaScript. The usage of this frontend focused technology is growing quickly. It is the 7th on Tiobe Index, which is a good indicator of programming languages popularity. [1]

Learning and teaching JavaScript, HTML and CSS is important. My tool focuses on this three main building blocks of the web.

Building a frontend heavy application, with a dynamic, user-friendly interface is more common nowadays. In the last few years JavaScript based frontend frameworks became mature, production ready tools. Server side technologies, like database management and time and resource heavy processes are separated from the user focused, design driven view layer, which is developed with usage of frontend frameworks.

The most popular tools are Angular.js, React.js and Ember.js. In my project I use Ember.js. It is an "opinionated" framework. Opinionated, convention over configuration driven framework means that developers should follow specific conventions, instead of using a tool freely. A more strict environment helps to adopt best practices and speed up the development process.

Certainly, we still have to store data and information, so we cannot live without backend and server technology. Luckily there are already cloud-based tools for managing databases. I use Firebase, which is a service provided by Google. Firebase is a cloud-based database, document-store solution and easy to integrate with Ember.js.

Additionally, I started to build a traditional backend server application also to sup-

port development and experimenting with a real server side and a cloud-based solutions parallel. My preferred technology on backend side is Ruby on Rails, a popular, also opinionated and convention over configuration driven backend framework. However, I added only one model to this backend, because finally I just focused the Firebase implementation.

Following the most modern standard of web applications, I separate the user faced frontend development and the data store, backend development.

The user face frontend application uses Ember.js frontend framework. Ember.js development requires Node.js on the development machine to run the development environment. This development environment helps to run and modify frontend code quickly, and it generates the final, deployable production code also. The production version of the application is only a static website. It means, there is one index.html, two JavaScript files and two CSS files.

3.2   Source control management

I have been using GitHub for managing source code and tracking code changes. Link to repository: `https://github.com/zoltan-nz/tutorial-builder`

3.3   Frontend features

The look and feel of the application follows the standard Bootstrap style. Bootstrap is added to the project. I use "sass" version of the Bootstrap, so I can customize it with the modification of the SaSS variables. SaSS is a modern CSS development environment, helps to programmatically modify the CSS.

The home page of the application is only a placeholder. I added a navigation bar with the following links: Home, Builder, Player, Sandboxes, Dashboard.

I implemented a breadcrumb bar also, which helps in navigation.

First, I created a Sandbox area, where I experiment with the CodeMirror code editor and an iFrame, which shows the preview. This Sandbox page contains the editor. When the source code is updated, the preview page automatically shows the generated website. This feature uses Ember.js default two-way bindings capability.

Database management is already implemented. The main adapter is the Firebase adapter, which automatically update data to the Firebase server. Firebase is a real

time database. The limited, free to use version is enough for experimenting and for demo.

The secondary adapter is a JSONApi Adapter. JSON Api [3] is a new standard of data communication format. This is the Ember.js default adapter. This active only when I use the Ruby on Rails based backend system, it is commented out in the production version, because I haven't added all model to the Ruby on Rails app. I keep it there, because the further development of this project will focus on this backend instead of using Firebase.

3.4 Static website hosting

I have been using surge.sh [4] for static website hosting, so the actual state of the prototype is updated there regularly. Link to the live version: `http://tutorial-builder.surge.sh`

3.5 Requirements

A fully functional tutorial service could have the following requirement list. We can separate them in three different groups. One group focuses on requirements for the tutorial creator/admin/teacher, and other group for the consumer/students and a separate group of requirements for user interface considerations.

Requirements from the teacher perspective:

- Teacher can navigate to Admin page.
- Teacher can create a new tutorial.
- Teacher can add steps to the tutorial.
- A step could be four different type:
    - Instruction type is a text content.
    - Html type, which adds content to the html editor box.
    - Css type, which adds content to the css editor box.
    - JavaScript type, which adds content to the javascript editor box.
- Teacher can modify the content of a step later.
- Steps always in sync, so the next step always inherit the previous step state.

Requirements from the student perspective:

- Student can see a list of tutorials.
- Student can click on a tutorial and can see the steps.
- Steps are presented in order.
- Student can "play" and "watch" the steps.
- Student can "pause" and step "backward".

User interface requirements:

- The tutorial screen has five area:
- Instruction area.
- Html code editor textarea.
- Css code editor textarea.
- Javascript code editor textarea.
- Html preview textarea.

The main website has two main section:

- Admin page where Teacher can edit tutorials.
- Tutorials page where Students can select and watch tutorials.

3.6   Design considerations

One of the core element of an application is the database and model layer. Determining data entities are an important step of the planning process and help us in the implementation process.

In a fully featured implementation our database structure would be more complex and would cover extended usecases. Firstly, I list a wider model structure, after I will focus on models what I implemented in the prototype.

Most of the web applications has a user and authentication layer, user related information would be stored in "user" model. Keeping it simple the user model would have three fields: username, password, is_admin. Username and password is the minimum what we need to identify different users. Additionally we have to store wheather the user has administrator privileges or not.

Because of our main problem domain is tutorials, one of the most important model is the entity where we store the different tutorials. So a "tutorial" model would have at least a "name" field. The tutorial model has many "lesson". Each lesson has a "title", a field where we can store the position, the "sort" value, and it has many

"steps" also. So the smallest level of this model structure is the "step" model. Step fields: name, type, code, sort.

Simply: Tutorial ->Lesson ->Step

The demo app uses this nested model structure in the "Dashboard" section. During the implementation process, my first iteration tried to cover this wider usecase, however, I realized, the challenge is not really on this area.

Finally I created a "builder" model and a "builder-step" model.

Builder model fields: name (string), builderSteps (hasMany)

BuilderStep model fields: builder (belongsTo), sort (number), comment (string), code (string)

With this simplified structure I was able to focus only the implementation of the tutorial and step builder functionality.

This graph summarizes these database tables and models:

USER	TUTORIAL	LESSON	STEP
username: string password: encrypted string is_admin: boolean	name: string lessons: hasMany	title: string sort: number steps: hasMany tutorial: belongsTo	name: string type: enum code: string sort: number lesson: belongsTo

BUILDER	BUILDER_STEP
name: string builderSteps: hasMany	builder: belongsTo sort: number comment: string code: string
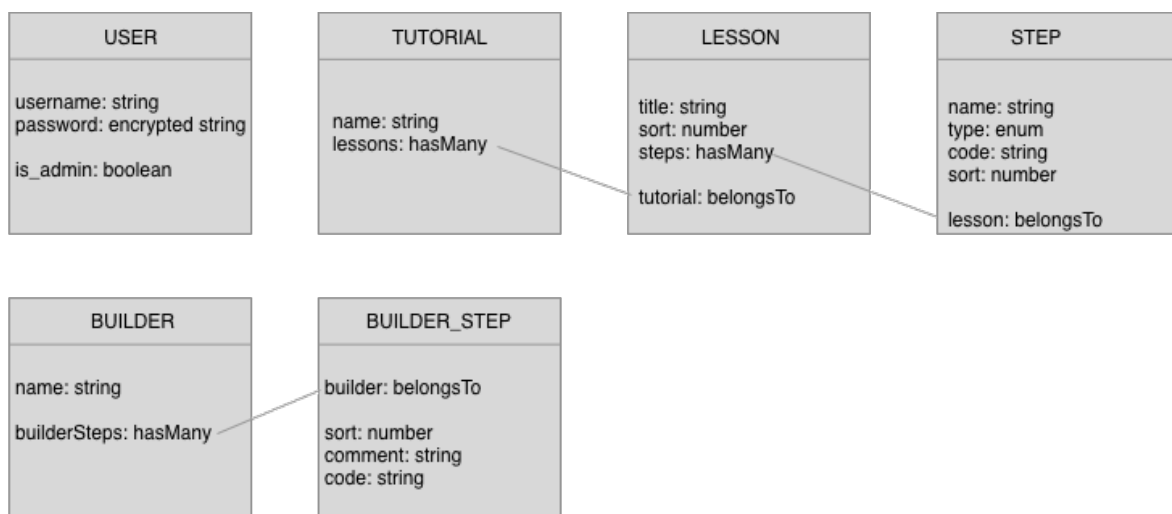
3.7   The story of 12 factor apps

The Twelve-Factor App is the name of a methodology, which collects together twelve important concepts and rules what we should follow if we build a modern software-as-a-service or web application. We can find this collection on this website: `http://12factor.net`

I think, it is important to follow this methodology in our application also, so I bring rules in my prototype implementation from this collection.

Instead of repeating all the 12 rules, I describe how I use a certain rule in my implementation.

1. Codebase. As it already mentioned above, our app uses one codebase, uses git and hosted on Github and the codebase is the same across all deploys.

2. Dependencies. External dependencies of our app managed by a package manager, in our case "npm", the node package manager and "bower", for third party assets, like Ember.js or Bootstrap.

3. Config. The configuration sits separately in "config/environment.js" file, and it could be different on development mode or on production mode.

4. Backing services. The Tutorial Builder backend system is attached via an adapter to the third party database, Firebase, which accessible via a direct URL. This distinct backing service is a resource.

5. Build, release, run. We can build our Ember application with a terminal command "ember build --prod", the production version of the code deployed by an other terminal command "surge" and it is released as production ready.

6. Processes. In our case this rule is relevant also, because our app is a static, single page application, so it is a stateless and "share-nothing" solution.

7. Port binding. This rule would be relevant only if I would use my own server. The backend service and the static single-page application would run from this server using a webserver. In this case the backend server, which could be a Ruby on Rails application, would run on a different port behind the webserver, for example on "http://localhost:3000/" address, however it would be open on the same domain name, but in a subdirectory. For example, if the our Ember.js single-page application would run on "http://tutorial-builder.com/", the Ruby on Rails backend API would be available on "http://tutorial-builder.com/api".

8. Concurrency. Our single page application is stateless, basically a static website. In terms of high traffic, it is easy to clone and launch on more server. Luckily, modern static website hosting services automatically clone it on content delivery network, which means, it can scale out.

9. Disposability. We don't have to turn off our web application, we can just deploy and overwrite the previous version in a second, so the next user will download the updated version.

10. Dev/prod parity. As a twelve-factor developer, it is important for me, that I should be able to deploy continously. It means, I can manage the code, update or fix it, test on the developer machine, test with the production database, build the production version and deploy immediately. With my development environment and with tools of my choice, this principal is adopted also.

11. Logs. This feature can be adopted in our environment, but it is not active in our experimental project.

12. Admin processes. Our database administration focuses only for maintaining, deleting the Firebase database, it can be done on Firebase website as a single process.

3.8   Data down actions up

In software architecture, and computer science it is always a challenge, how could you write clean code [5], which is always readable and easy to maintain. I built my project mainly in JavaScript, and it is known that JavaScript is not so strickt language, no strick types as in Java, it doesn't force structured object oriented patterns. However, JavaScript changed a lot in the last few years, thanks for the new version. The new JavaScript standard called ES2015 or ES6 [6].

JavaScript with ES6 syntax and with a heavily object orinted JavaScript framework, like Ember.js, can force you to write an easy to understand, easy to maintain system. Ember.js is a Model-View-Controller type framework and it helps to separate concerns and make your code more SOLID [7].

With modern JavaScript frameworks we mainly build frontend, user faced applications, so the view layer of our product is a webpage or a web component. These pages or components are a mix of static and dynamic content. Static content is built in the presentation template, however the dynamic content usually provided by a backend service from a database.

Modern JavaScript frameworks are adopted a new pattern, which determines the data flow inside the application. It is driven by the user interaction. When a user opens a web application and navigates inside this webapp, usually the browser changes the actual web address in the location bar. We call it "routing". So when the route changes, mainly we navigate to a new page. We usually call it, the "state" of the application is changed. It triggers a series of steps. One of the most important step, that the app tries to access to dynamic data which will be rendered on the page. This data can come from an allready cached local data store or can send an AJAX request to the backend system and download it. After this data rendered on the page.

On a website the previously prepoluated data can change, for example in a web form we change a text in the text field, or change a code in a web based code editor, or the user click on a button, which turns on or off an information. Each case we actually change the data.

Originally web frameworks are introduced a uniqe concept, which is called two-way bindings. When data change somewhere in the application, it broadcasted everywhere, so quite quickly we were able to build a dynamic web application. In our project, when a user edits code in the code editor, the preview window automatically updates, because the code is change in that IFRAME also.

Two-way binding has own benefits and use cases. It is very usefull when we use data in the same context (same page, same object oriented class in our code). However, two-way bindings can cause data leak, so when we change something on the page, an unexpected behaviour could happen. It can be easily managed in a small application, but when our application grows, very difficult to maintain.

For this reason, there is this new pattern, what we call "data down actions up". It is prefered to turning off the two-way binding in the application and keep the automatic update in a small scope only. When we want to populate the new data, it could be calling with a direct function in our app. It is called actions. So instead of letting the invisible "magic" update states in our application, we clearly call and send actions to the other part of the app.

The benefit of this, if someone else try to understand our code, there is not any invisible part.

3.9   Low level design decisions

In this section I collect together those questions what appeared during the implementation process.

3.9.1   Managing css with sass and adding bootstrap.

Bootstrap[8] design system is still one of the most popular, out of the box solution when we need a decent, desktop and mobile friendly layout and UI/UX.

Bootstrap originally uses Less[9] css compiler, however the most popular nowadays is Sass[10]. (Css compilers help us to maintain a larger stylesheet system, so it can keep our css structured) I prefer to use Sass with Bootstrap.

Ember.js is famous about its add-on ecosystem. We can extend our project with a one liner console command, we don't have to manually download, copy-paste files in our project, enough if we run a simple "ember install" and this command line tool manages it for us.

I use this ember installer for adding Sass and Bootstrap to our project.

Bootstrap has a quite large variable file, where we can configure and customize the look and feel. This is placed in the 'app/styles' folder.

During the project I added a few extra css class which helps to highlight the code preview window.

3.9.2   Code mirror plugin

After reviewing web based code editor options, I choosed Code Mirror, because it is widely used, well documented and easy to add to our project.

Ember installer can help us in this case also. It adds with a one line command the default Code Mirror library and settings in our application.

However, turning on extra features and setup different colorschema on this plugin involved more customization.

I followed the documentation on the Code Mirror official website and the Ember Addon website. Unfortunatelly, the documentation of the Ember addon was not correct, so I had to manually disassemble the source code and figuring out how can I add the configuration what I planned. Thanks for this additionaly changes, the code editor has the "solarized" style, it can manage "vim", "emacs" and "sublime" key maps in my project. The editor can understand "xml", "javascript", "css", "handlebars", "markdown" syntax.

I wanted to extend the editor with the widely used "Emmet" feature. Emmet is a code editing mode of html and css and it automatically completes your code, when typing. One of the real challenge was to implement this addition, I write more about this in the Challenges section.

3.9.3   Breadcrumbs

One of the most important rule in UX, that the user should not be confused. When they use a website and navigate in different arrea, they have to know where they are. Usually under the navigation bar, there is a row, where we show the actual

16

position inside the site architecture. It is called breadcrumbs.

Adding breadcrumbs to the project started with a search. We can find Ember.js addons on `www.emberobserver.com`. The most popular breadcrumb addon is ember-crumbly [13].

With the "ember install" command line tool, this package is added to the project. I was able to insert a bootstrap compatible breadcrumb under the navigation bar, which automatically update when we navigate to a new page.

Screenshot about the implementation:
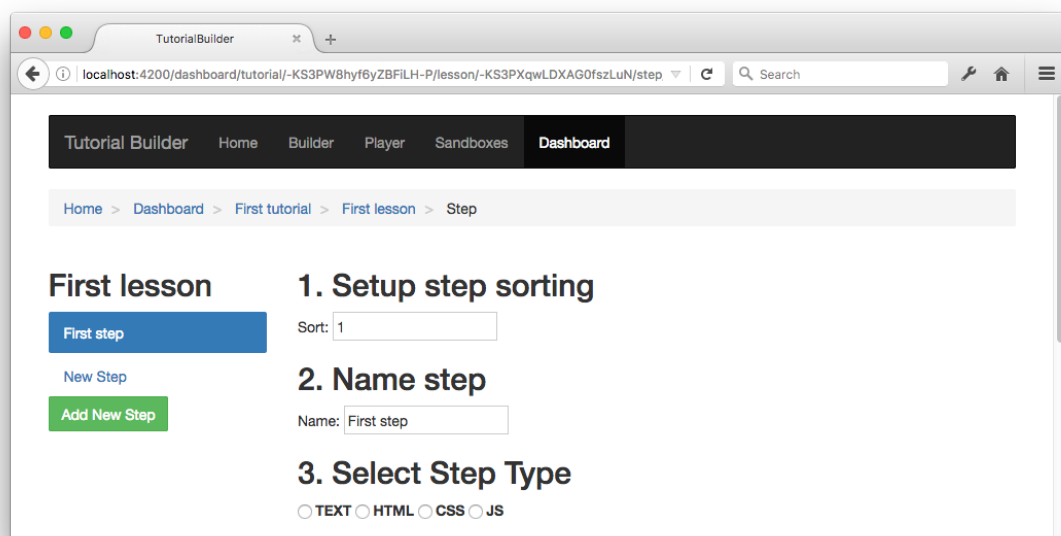
3.9.4   Communication between two components

There is an interesting problem domain on JavaScript applications. When we build a JavaScript based single page application, we use little elements to build a larger section. This little elements are called components and moving this components next to each other create a webpage. Components reusable, so we have to create, develop only once and we can insert in different pages. The above explained breadcrumb is a component also, which was created by other developers, so we can just simply insert in our app.

However, these components usually independenlty do what they has to do, and they don't worry too much about other part of the application, except if we connect them together.

Other problem with the components, that maybe a little part from a component should be inserted somwhere else.

Luckily other developers already solved this problem. I added "ember-wormhole" addon to the project, because this helps me to insert logically connected html elements somewhere else on the website.

In my step editor, there is an "Undo/Redo" button component, which belongs to the Step editor form, which is on the right side of the page. This wormhole plugin helped me to keep these two part connected.

You can see the implementation in this screenshot:
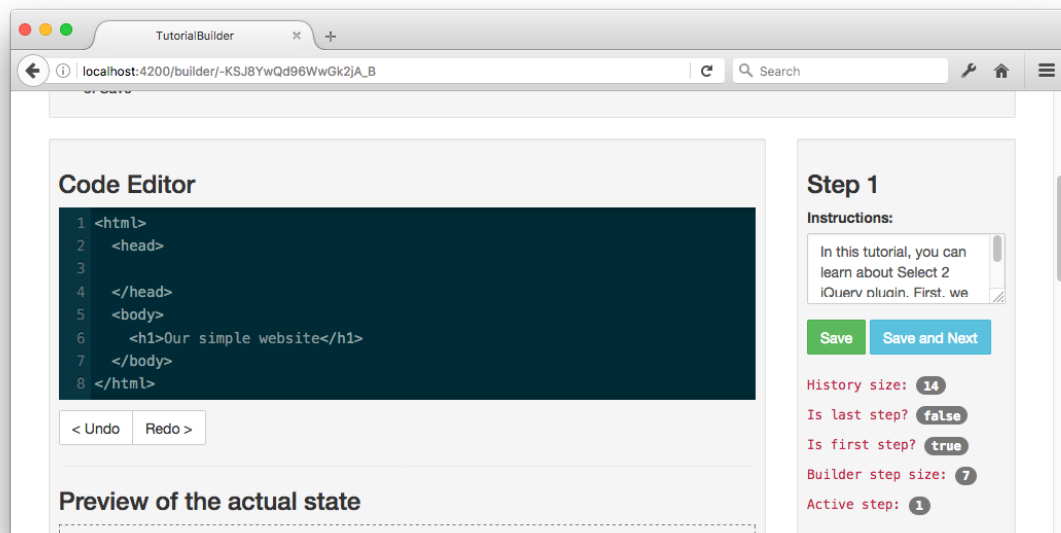
3.10 Challenges

3.10.1 Addig Emmet to Code Mirror

Emmet is an essential toolkit for web-developers, which can be added to most of the popular code editor. We can write html and css code only using simple abbreviation [11].

As I mentioned earlier, using Ember install helped me to add the Code Mirror code editor. Unfortunatelly, this addon focuses only the very basic user scenarios, mainly

for simple code editing.

Digging deep in the Ember Addon implementation, which is called Ivy-CodeMirror [12], I found out how the Code Mirror instance managed by this addon.

In my codebase, you can find a unique special component, which initialize Emmet with CodeMirror, when a code window added to the page. (Location of the component file: "app/components/ivy-codemirror.js")

Thanks for this changes, in my final implementation you can use this Emmet editing feature in each editor window.

3.10.2   Database management

Managing database and choosing the right solution is always desire more research and more iterations.

My project finally use Firebase, as I planned in my proposal also, however I had to try out other options also. During the development process, I had problem with the speed of the Firebase.

Firebase is a cloud based database, so it needs internet connection all the time. We cannot use the app offline. Additionally Firebase has limitations also.

Firstly I tried to build a unique backend system in Ruby on Rails. You can find the actual state of this application on my Github [14]. I implemented a special saving mechanism in the code editor. When we modify the code in the editor, the code changes will be saved every third second automatically, so it does not overheat the database.

However, implementing a full backend system is a big overhead in a smaller prototype project, like this.

Secondly I tried to use a mocking system, which is popular in Ember.js ecosystem. This mocking addon is called Ember Mirage [15]. When you use a mocking system in a frontend application, you can run the application without any backend support, because all the database related backend respond is generated by an other JavaScript code, a mocking system. You can read and save data. It is super fast and easy to experiment with it, the only disadvantage, that the saved data is not permanent, if you close the web browser, all data will be deleted. Usually we use "fixtures" for prepopulate the database when we open the app with a mock database. I experimented with this mocking system, but finally I went back to Firebase, because it was important for me to upload the app to a public server and connect it to a real database, so

I just continued the development process with Firebase.

3.11   Managing history changes

In a step by step tutorial builder, one the core functionality is adding steps to a tutorial. It is important also, that we should be able to edit, modify, delete, reorganize these steps.

A programming tutorial main content is code. Code has lines. Usually in the first step we add a few lines, so there will be two-three lines of code in the code editor. In the next steps we add more and more. We extend lines, remove lines. Each step based on the previous state, each incrementally change the code in the code editor.

Let say we have 4-5 steps and we would like to remove the code what we saved in the Step 3. After this changes the Step 2 should be the start state of the Step 4, so we should manually adjust Step 4 start state to be the same as the end of Step 2. We can show the difference between the final state of the Step 2 and the actual start state of the Step 4, highlight the differences and ask the user to adjust. This looks simple when we talk about a few lines of code. However, when we build a bigger tutorial, this could be a nightmare.

I was thinking about this problem a lot and I tried to use the following simplification to solve this problem.

One of the idea is creating a snapshot at beginning of a step and at the final state of a step.

In the following abstraction, H1, H2... is a history element, something what we do with the code. S1, S2... is the Step. L1, L2... is the line of code.

- H1. Snapshot N1, Empty state"
- H2. Create Step S1, add Line L1
- H3. Snapshot N2, H1 + H2.
- H4. Create Step S2, add Line L2
- H5. Snapshot N3, H3 + H4
- H6. Modify Step S1, add new Line after L1
- H7. Update Snapshot N2: H1 + H2 + H6
- H8. Update Snapshot N3: H7 + H3

In this concept a snapshot like a pointer, which collect together the connected history elements.

However, this concept could work only if all the changes what we make in the code editor saved as relative changes. For example, if I have something in the Line 1 and we add something in the Line 2, I save the changes as "add a line after the last line of the previous step", so if we delete the Step 1, the relativ instruction still work.

Unfortunatelly, managing relative history is not possible with Code Mirror at the moment. My plan is to extend the tool with this feature in a following project.

I was thinking about an other theory. We use git and git main feature is tracking changes. It keeps the differences between two commit, so we can see what's happened between two commit. These changes stored as relative changes also. However, one of the strict restriction in git, which is an important feature also, we cannot easily modify the history of the commits, so when would like to rewrite the history, we actually has to make a copy from the actual git repository.

In this case, the problem is the same again as I described above. How could we match together the previous step final state with the next step start state. Git raise a "conflict", so the user has to manually resolve this problem. So we are on the same path again with this theory also.

3.12   Reflection

3.12.1   Great part

I think, it was great working on the architecture of this project. Thinking about the different element and technology and how these parts connect together. Working with Ember.js is fun and I learned a lot about this modern, matured frontend framework.

Bulding the backend system, and trying out different approach was interesting, especially using Ruby on Rails, which is one of my favourite technology.

I am impressed, how complex and advance most of the web based code editor tool. Reviewing them gave me the opportunity to understand more, how they work. Adding Code Mirror to my project was a good joice, because it is feature heavy with an extensive public programming API.

3.12.2 Need improvement

As I already explained in the challenging section, managing history, deleting or moving steps in a tutorial could be improved.

3.12.3 Would do differently?

This project is about experimenting and learning new tools, solving problems and developing solutions for a certain problem. This process is obviously involve steps which does not produce real value at the end, however, it can proof that direction is not the right way.

I feel, my plan was too ambitious at the beginning, and I had to realize that not so easy to implement what I imagined.

Chapter 4

Evaluation

4.1 Process

The product evaluation has two parts. The first part is expert evaluation, which means, I collected feedbacks from my direct colleagues (from my workplace and from research group). In this section I describe the implemented product with screenshots. The second part is a survey, which helped to collect opinion about programming tutorial usage.

4.1.1 Expert evaluation

Expert evaluation helps to discover quickly the "low-hanging fruits", obvious usability problems, without investing extra resource for a wide user testing.

My reserach product is a prototype, which cannot be perfect for the first iteration, but already can help us to direct our future study.
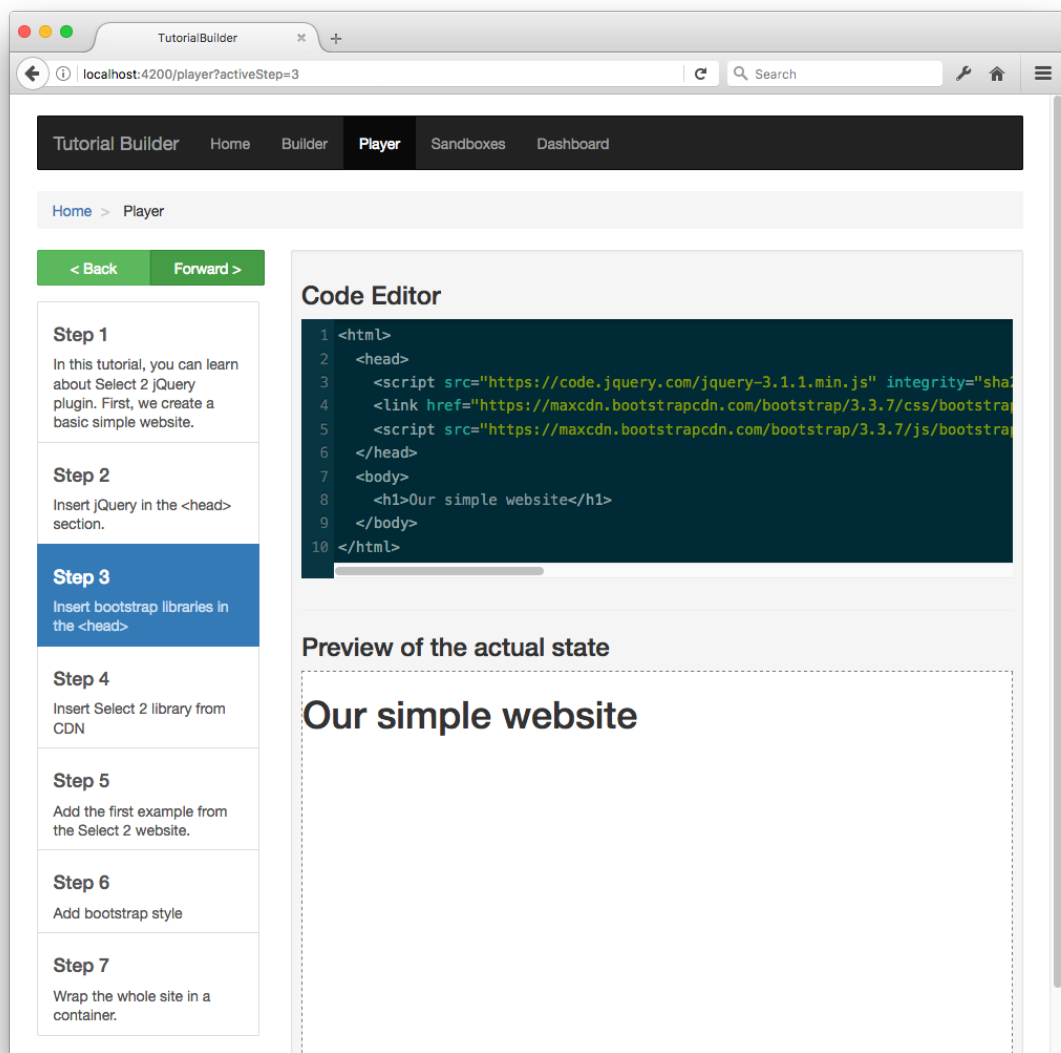
How the app works and what it does?
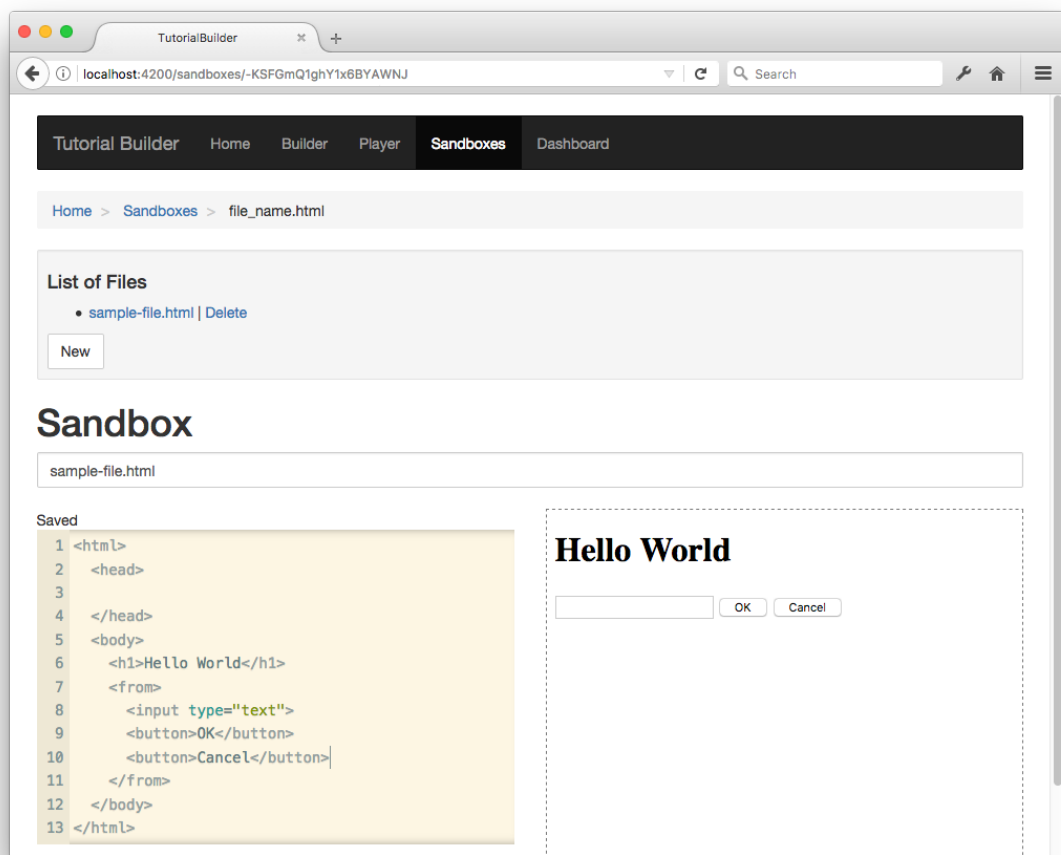
Open screen is mainly just a placeholder.
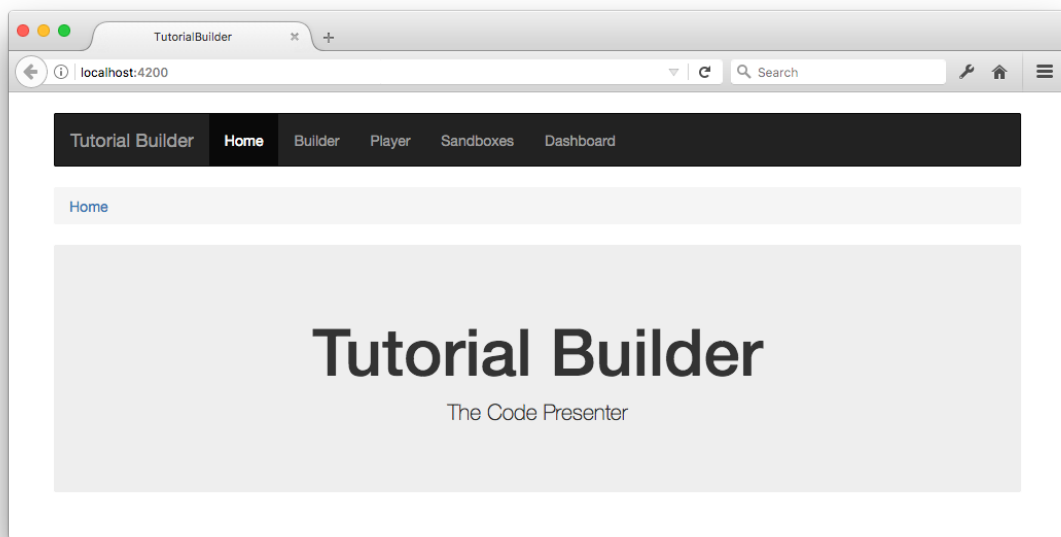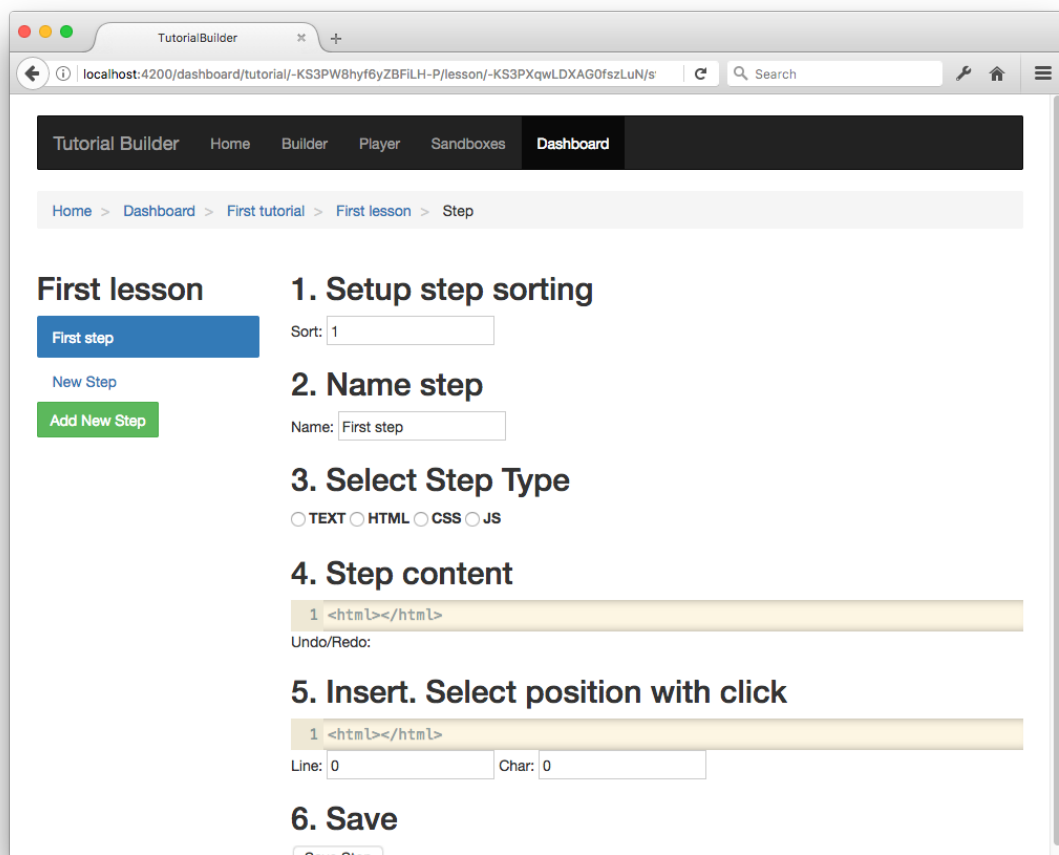
The Builder
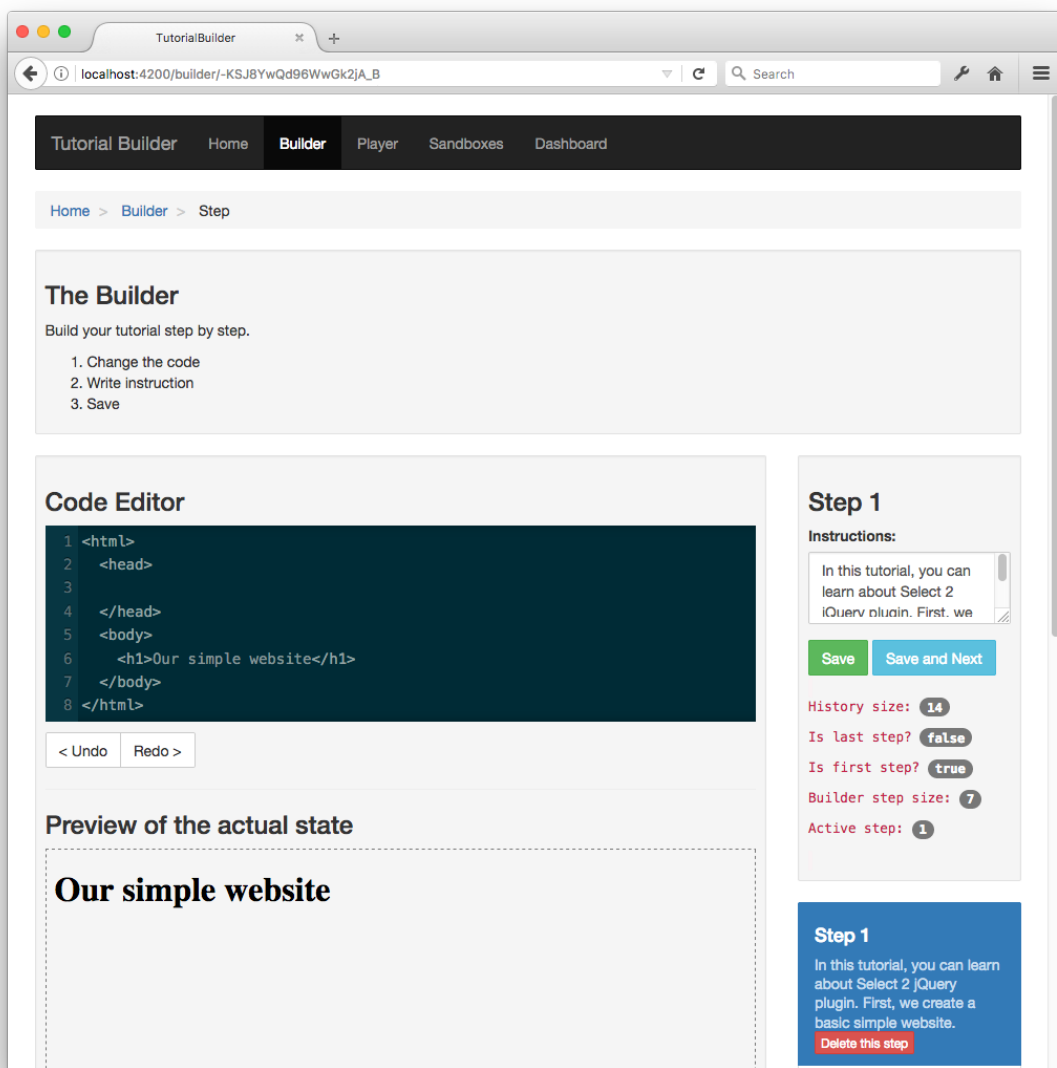
The Player

Sandboxes

Dashboard

4.1.2 Tutorial Builder Survey

Watching, reading or following tutorials are important part of our learning process, but everybody could have different motivation. I launched a survey for collecting information about user's motivation and their goal.

Victoria University ethics committee approved my request, so I run a little campaign on my twitter, on my blog and shared the link in different developer chat groups.

I built the survey on Google Survey platform, and it was presented as the following screenshot shows.

Survey About Interactive Tutorials

Hi, I'm Zoltan, and I would like to ask a few questions about programming tutorials. I'm studying on Victoria University Wellington, and your answers would help in my research project. This is an anonymous survey.

When you learn a new framework or experiment with a new library/tool, what is your main motivation?

☐ Need for my work, it is part of my job.

☐ I have a hobby project and I would like to solve a problem.

☐ I just love exploring new technologies, it is fun.

☐ Other: _____

How experienced are you in programming?

○ Beginner

○ Intermediate

○ Experienced

What is your prefered method to get to know a new framework/library?

○ I prefer to read documentation and to see static code examples

○ I like to watch videos about the framework/library

○ I prefer to try the framework/library in an interactive code editor and to see the preview

○ Other: _____

The questions and options:

When you learn a new framework or experiment with a new library/tool, what is your main motivation?

- Need for my work, it is part of my job
- I have a hobby project and I would like to solve a problem.

- I just love exploring new technologies, it is fun.
- Other...

How experienced are you in programming?

- Beginner
- Intermediate
- Experienced

What is your prefered method to get to know a new framework/library?

- I prefer to read documentation and to see static code examples
- I like to watch videos about the framework/library
- I prefer to try the framework/library in an interactive code editor and to see the preview

When you try a library or a framework in a web based code editor, which features are important to you? (Scale from 1-not important to 6-very important)

- There is a step by step introduction of the usage of the library
- I can pause, reverse the introduction
- I can modify the code in the web based code editor
- I can share/export/copy the code from the web based code editor

What other features would you like to see in a web based interactive tutorial. (open question)

I got 79 answers.
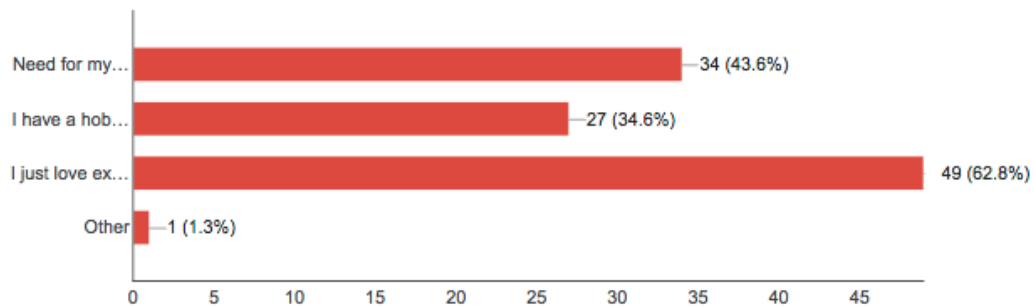
The results from the survey.

79 responses

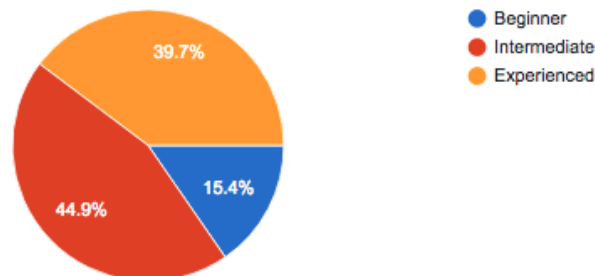SUMMARY   INDIVIDUAL

Accepting responses

When you learn a new framework or experiment with a new library/tool, what is your main motivation?
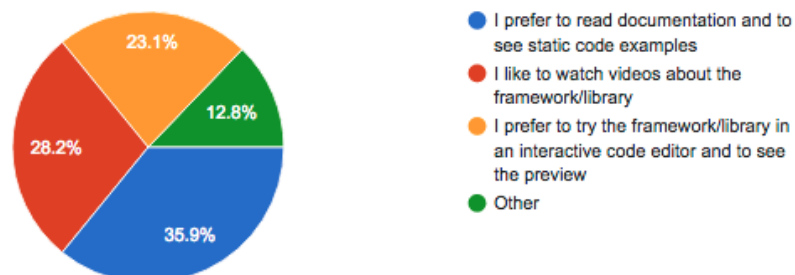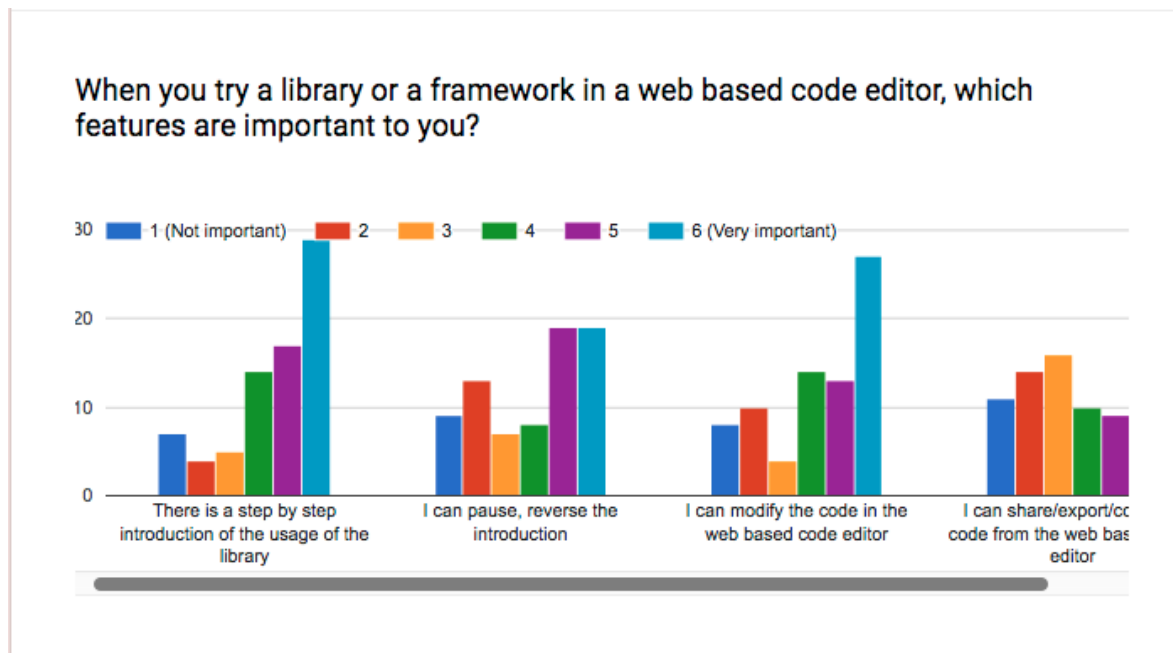(78 responses)

Need for my... — 34 (43.6%)
I have a hob... — 27 (34.6%)
I just love ex... — 49 (62.8%)
Other — 1 (1.3%)

0   5   10   15   20   25   30   35   40   45

How experienced are you in programming? (79 responses)

- Beginner
- Intermediate
- Experienced

39.7%
15.4%
44.9%

What is your prefered method to get to know a new framework/library?
(78 responses)

- I prefer to read documentation and to see static code examples
- I like to watch videos about the framework/library
- I prefer to try the framework/library in an interactive code editor and to see the preview
- Other

23.1%
12.8%
28.2%
35.9%

When you try a library or a framework in a web based code editor, which features are important to you?

It was great to see how many suggestions arrived for the open question.

Most importants:

... ...

4.2 Analysis

Working on a project like this, helped me to experiment with different solutions and I learned a lot how can I connect together different part of a more complex solution.

The step by step tutorial is just a beginning. I visualized a much bigger project, which actually can be really helpful for teachers and for students also.

My original plan was to show how could you build a more complex application. When we build a web app, we deal with a bunch of files and complex directory structure. So many moving parts.

Keeping this in sync, and showing the changes of directories and file content, we already use Git. So one of the obvious choice could be to bring together a step-by-step builder tool, like this prototype and instead of storing this steps in a database, we should use a git repository website, like GitHub or Bitbucket for storing states, steps. Each commit would be a step. It clearly shows the difference between actual

step and previous step.

A git repository store the file structure and file content, using the api of this service, we can show on our website, in our code editor, the whole project.

The challenge is the same as in this little prototype. How can we go back in time and change an earlier step, with keeping the history.

With git, we can add a new commit to the history, sequentially, however, it would modify an interim step. This is really a challenge. Because the code change history is in series, we cannot modify that one, however, we can reference back to the original step, and actually the change is not really the codebase change, it is more likely the step change.

Chapter 5

Conclusion

Bibliography

[1] Tiobe.com - `http://www.tiobe.com/tiobe_index`

[2] CodeMirror Movie - `https://github.com/sergeche/codemirror-movie`

[3] JSON Api - `http://jsonapi.org`

[4] Surge.sh Static Website Hosting Service - `http://surge.sh`

[5] Book: Robert C. Martin - Clean Code - ISBN-13: 978-0132350884

[6] ES6 Standard - `http://www.ecma-international.org/ecma-262/6.0/`

[7] SOLID pattern - `https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)`

[8] Bootstrap - `http://getbootstrap.com`

[9] Less - `http://lesscss.org/`

[10] Sass - `http://sass-lang.com/`

[11] Emmet - `http://emmet.io/`

[12] Ivy-CodeMirror - `https://github.com/IvyApp/ivy-codemirror`

[13] Ember-Crumbly Addon - `https://github.com/poteto/ember-crumbly`

[14] Tutorial Builder Backend - Ruby on Rails app `https://github.com/zoltan-nz/tutorial-builder-backend`

[15] Ember Mirage - `http://www.ember-cli-mirage.com/`