Interactive Web Presentation for FactCheck framework

Practical Course: Computer Science 1

Hynek Zemanec*

Faculty of Computer Science, University of Vienna, Währinger Str. 29, 1090 Vienna, Austria

2022S

Contents

Abstract

The project aims to aid the introduction of the FactCheck framework to the newcomers using an interactive web presentation. Areas of concern are explanation of important vocabulary, animated visualizations of foundational concepts, overview of underlying FactCheck services and their architecture, showcase of existing projects and their authors, offering of the available projects, interactive demonstration of the framework and a point of contact. Leveraging the rich ecosystem of the JavaScript library React, state of the art React framework NextJS is configured to support dynamic routing and the build process producing static exports. Design system Material UI, with styled JSX, modified to support SASS syntax is used to ensure visual consistency. TypeScript was used to ensure type safety. Additionally, customized NPM scripts together with GitLab Pipeline configuration for automated deployments is provided. Repeated user testing ($n = 10$) were basis for content improvements and identification of usability issues.

*e-mail:hynekz20@univie.ac.at

1 Introduction

FactCheck is a cloud-based framework for automatic detection and resolving of conflicting data on the internet. It is developed by the Multimedia Information Systems research group at the University of Vienna. The framework consists of a large number of modules and individual services for different aspects of the framework such as data acquisition, data comparison and analysis, presentation of the results and other related functions. The idea behind the framework is not to be an arbiter of factual information. The goal is to provide an overview of possibly conflicting information based on the same information found elsewhere on the internet. This tends to confuse new comers. Consequently, large time investments are needed to give a proper introduction into the idea of the framework as well as interpretation of the underlying complex processes. Facilitating the onboarding of parties interested in FactCheck framework, this project aims to mitigate outlined difficulties by means of an interactive web presentation.

2 Project Definition

The main goal is to introduce main concepts in layman's terms in visually attractive and intuitive setting. Successful execution of the project mandates sufficient high-level understanding of framework's components and respective projects. The implementation is carried out using multimedia elements facilitating interactive storytelling via logically connected sections.

Sections provide a seamless integration (both content-wise and by means of fluid transitions), resulting in a default story path concluded by a call to action. Furthermore, the online experience is tailored so that user can choose to navigate through the sections in arbitrary order without loss of context. Supporting the idea of an uninterrupted experience, the website has been conceptualized as a one-page website with dynamically animated parts triggered and loaded by scrolling. Since the website has been built using a mobile-first design approach, the advantage of web based solution is that it can be easily viewed by any device with an access to the internet, with a modern web browser installed.

For the sake of completeness, the original description of the project as offered by the department reads: *This topic aims to create a convincing, clear, understandable presentation of the FactCheck framework as well as its related components and services. The result of this work is an interactive website that makes use of illustrations, animations, and other interactive tools to present FactCheck in its full range. The interactive setup should allow for a personalized experience, guide the user through the topic, and therefore provide a better understanding of complex interrelations.*

2.1 Concept

In the initial phase of the project, conceptual map[1] has been modeled to capture all areas of concern to be addressed. Further broadening the range of requirements suggested earlier, the following list enumerates identified concerns in the conceptual map:

- What is FactCheck framework
- The problem setting
- Vision and the idea of the framework
- Description of the process that arrives to the comparison results
- Architecture of the FactCheck framework
- Showcase the ongoing and finished projects
- Advertise the available projects and technologies used
- Introduction of the team
- Live Interactive Demo
- Point of contact using form and available contact information

Beyond the identified requirements, adequate multimedia constructs as well as conceptual illustrations are included in the map. During the iterative process of merging related parts into logical sections, a following sequence of sections emerged:

1. Introduction: Landing Page with simple illustration
2. FactCheck Journey: Interactive slideshow describing steps
3. Gallery: Screenshot previews from implemented projects
4. Architecture: Illustrated overview of FactCheck constituents and technologies
5. Demo: interactive editors communicating with FactCheck API
6. Projects: description and techstack of all past, present and available projects
7. Challenges: identified shortcomings and opportunities for development
8. Team: some team members and their bios
9. Contact: contact form

2.2 Graphical Prototype

Since the conceptual map has substituted the role of wire-frame diagrams, the next phase was a graphical design. Thanks to source files provided by Material UI library for variety of prototyping software, the graphical design is visually consistent with the final website. The prototyping software Figma[2] was used both for the reference design as well as production and export of vector graphics.

3 Project Management

The project management has mirrored the philosophy of agile development. Specifically, with the exception of the design and conceptual phase, the development has been iteratively carried out in weekly sprints, with gradual improvements being made based on received feedback. Such weekly sprint can be generalized as:

1. Gather requirements/feedback
2. Design
3. Implement the changes
4. Present and test

Milestones and sequence of activities are illustrated by the the prospective schedule in table 1.

[1] Miro Board available at `https://miro.com/app/board/uXjVOCi1HyO=/`

[2] Design available at `https://www.figma.com/file/1FjSXEqwC1WBOViOEHFsP6`

3.1 Data Management

As previously mentioned, one of the tasks was to showcase existing projects, offer the available ones and introduce areas with potential for improvement. This required understanding of the constituent FactCheck projects and contacting of the concerned authors to verify description correctness. The process of producing comprehensible descriptions of individual projects lead to an organized collection of 14 definitions of projects together with their author contact information and their current status. The file is stored in `\fact-check-landing\public\data\projects.json`. Additions to the file is reflected in the projects sections, where each `project` is rendered as a component. Moreover, projects can be filtered by means of the implemented filter component, narrowing the selection based on the project status. Listing 1 demonstrates an expected structure of a project entry.

Listing 1: Project format

```
[
  {
    "author": "John␣Doe",
    "project": "Example␣Project",
    "mail": "example@univie.ac.at",
    "techstack": ["Python", "Azure",
        ↪ "UML"],
    "status": "Done",
    "description": "Example␣Description"
  },
...]
```

In the same location as projects, a JSON file `challenges.json` is stored, listing current shortcomings of the FactCheck framework. Similarly to the projects, entries are processed and are dynamically render into a word cloud. The relative size of each entry can be controlled using `count` parameter accepting values from the interval $[1, 50]$. Listing 2 demonstrates an expected structure of an entry.

Listing 2: Challanges format

```
[
  {
    "value": "User␣acceptance␣testing",
    "count": 38
  },
  ...
]
```

3.2 Version Control

The source code has been version-controlled (and remotely archived) by the GIT platform GitLab provided by the University of Vienna. The code is available online at `https://git01lab.cs.univie.ac.at/p1/2022ss/12010957-hynek-zemanec`. Top-level `README.md` contains instruction for local development. The git workflow used for versioning management utilized `issue#-dev-master` branching model where:

1. `issue#` refers to the branch associated to given issue number allocated to it using the GitLab issue tracker
2. `dev` is the development branch
3. `master` tested code released to the the production

In this model issue branches are made from the development branch that contains the latest changes. The issue branch after being completed is merged back to the development branch which is then merged to the master branch. Merging with the master branch triggers automated deployment process.

3.3 Deployment

The deployment takes advantage of the GitLab's CI/CD pipelines. The configuration of the stages of the automated deployment process is defined within `.gitlab-ci.yml`.

1. build
2. deploy

As the name suggest, the build stage transforms TypeScript files together with all dependencies into a single bundle of statically generated files. This is achieved by downloading the image of node and running the the NPM scripts `yarn` and `yarn export`, to download and install node dependencies and generate static files (as a result of the build process), respectively. The deploy stage pulls an image of Ubuntu, installs an SSH agent and using a provided SSH key synchronizes the artifacts of the build stage with specified remote directory. The process is depicted in the UML state diagram in the figure 1.

For hotfixes and other manual deployments, the developer can run `yarn deploy` which runs the same series of commands except for last step calling a shell script `deploy.sh` which synchronizes generated files with the server using the linux utility *rsync*[3].

4 Architecture

The main state of the art tool used in the project is the React framework NextJS. The framework provides abstraction for many different aspects of web development that are not part of the core React library. Examples of these abstractions are routing, CSS in JS support, image optimization, TypeScript support, server side rendering, static generation and others to name just a few. Although not required, NextJS and other similar libraries tend to group both business logic and style information in one file. This may arguably break the principle of separation of concerns, however it proves beneficial when targeting elements within an isolated context of a component. Since NextJS offers support for TypeScript, static type checking is offered

[3]This generally requires a password or SSH key for server authentication

Table 1: Prospective Schedule

WEEK(s)	Duration (weeks)	Milestone
1	1	CONCEPT: Gathering and Modelling of Requirements
2	1	CONCEPT: Wire-frame Prototype Design , feasibility experiments
3-4	2	IMPL: Initial prototype
5-9	5	IMPL: Implementation, Design Iterations & User Testing
10	1	IMPL: Final Modifications & Deployment
11	1	REPORT: Finalize the Project, Begin Report
12-13	2	FINAL: Finish report, Present Results, Lessons Learned

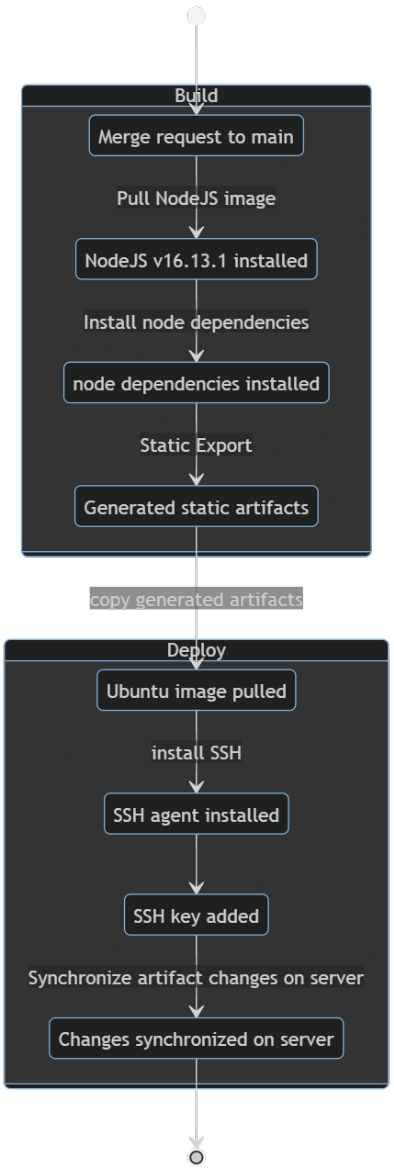
Build
Merge request to main
Pull NodeJS image
NodeJS v16.13.1 installed
Install node dependencies
node dependencies installed
Static Export
Generated static artifacts
copy generated artifacts
Deploy
Ubuntu image pulled
install SSH
SSH agent installed
SSH key added
Synchronize artifact changes on server
Changes synchronized on server

Figure 1: CI/CD State Diagram

by the editor, ensuring type safety. Additionally, TypeScript provides automatic documentation within the editor and therefore the project can be easily modified or expanded.

React allows developers to build user interfaces in a declarative fashion. In general, it follows the object-oriented principle of *composition over inheritance*, that manifest in a recursive component structure where React translates to JavaScript functions written using XML-like JSX syntax. The main shortcoming of this approach is complicated management of shared state[4] that needs to be passed through the chain of components and be kept at the highest level component (Higher Order Component or HOC). To enable better scalability, developers tend to use global state management solutions such as Redux or React's Context API. Since the application does not utilize shared state, the solution via HOC is sufficient.

4.1 Directory Structure

The directory structure of the project source directory `fact-check-landing` is as following:

```
/
└─ fact-check-landing
   ├─ components
   ├─ configs
   ├─ layout
   ├─ pages
   ├─ public
   ├─ sections
   ├─ slides
   └─ styles
```

The `components` directory stores component modules that are meant to be reusable pieces of code contained in pages. Configurations of the NextJS background processes are stored in the `configs` directory. Layouting elements used in pages that semantically and visually group components are located in `layout`. Aforementioned directory `pages` stores pages which are top-level components components that expose URL paths that are identical to their name[5].

The folder `public` contains static resources that are meant to be kept unchanged and merely copied to the

[4] state where n components is subscribed

[5] E.g.: a file `projects.ts` translates to a route `example.com/projects`

production bundle (images, server configurations, manifest etc.). Sections located in the directory of the same name are logically grouped units of content used withing pages. A specific type of directory is `slides` in which components that have contents and behaviour specific for a slide used in a `Presentation` component are located. Finally, `styles` directory contains configuration files for the material design system and globally utilized CSS styles. For an easier understanding of the relationships between described constituents, figure 2 illustrates their composition hierarchy.

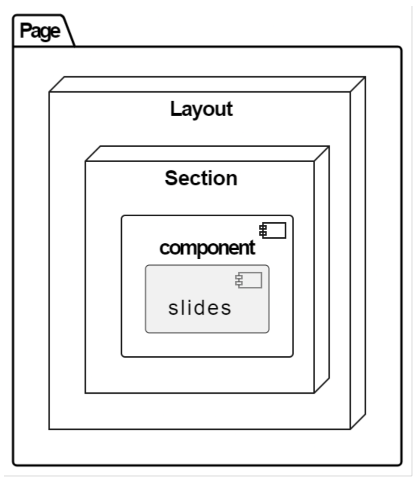

Figure 2: Component Hierarchy

An entry file for the application located in the pages directory is `_app.tsx`. Here all essential components that are required to be passed down through the component hierarchy are imported. This includes fonts, theme settings, style resets, layouting logic and possibly other higher order business logic.

4.2 Security

At the time of writing of this document, the website demo is available at `https://wwwlab.cs.univie.ac.at/~hynekz20/fc/`. Since the website is not meant to be publicly accessible, it is protected by simple authentication using combination of `.htaccess` and `.htpasswd`. Passwords are hashed by cryptographic algorithm SHA-1. Both files are located in the previously mentioned `public` directory and are valid for configured Apache2 HTTP server. For access credentials, contact the author or a MIS research group member.

5 Visual Elements

For visual consistency, the project uses Material UI (MUI) Design System library that offers variety of visually pleasing, accessible and optimized components. This also covers fonts and font sizes, colors, shadows, viewport breakpoints, transitions and other elements frequently encountered in the build process. Using a design system ensures identical behaviour and appearance of the website across browsers. Moreover, the design system used bases it's existence on Material design system developed and maintained by Google. This provide assurances that components have been extensively tested for usability, intuitiveness and that they conform to industry best practices. Configuration of MUI library can be found in `\fact-check-landing\styles\theme.js`

5.1 Layout

The semantic layer of the layout is defined as 3 high-level regions:

- Header
- Main
- Footer

In header, navigation bar is placed with links to essential parts of the page that have been evaluated as the most important. Those are:

- How It Works (presentation of main concepts)
- Demo (interactive editors communicating with API)
- Projects (grid of project descriptions)
- Contact (contact form)

Main region is the largest container that encapsulates all content that is further subdivided into sections. The entry point for individual sections is located in the `index.ts` in the pages directory. Finally, footer contains a contact form and auxiliary website information. Described structure of the layout is advantageous for introduction of new pages, where the main content can be easily changed and footer with header are kept unmodified.

On the lowest level, the layout is managed by the flex-box based Grid system of the MUI library. The library defines a *container* component with *item* children that can have specified fracture of horizontal space occupied within the container depending on different viewport sizes (*breakpoints*). The breakpoints in px units have been defined as the following:

- `xs = 0`
- `md = 600`
- `lg = 900`
- `xl = 1200`
- `xs = 1536`

Breakpoints ensure that valid `media-query` rules are generated and applied automatically to facilitate a responsiveness of components. For projects and other grid-like sections the `Masonry` component is used, that resemble a pattern of an exposed brick wall. This means that components are dynamically placed in the grid based on the free space available.

5.2 Animations

In the project 2 approaches are used regarding animations and element transitions. The first uses built-in transitions in the MUI library that are applied uniformly to the Components and the second approach uses CSS animations targeting DOM elements. The former is used for dynamic transitions of components such as sliding, fading, growing or zooming in. The latter has been used for more granular control over complex interactions. To animate the part of the web page when user scrolls it is required to keep track of what component is currently in the viewport. This is accomplished using the *react-intersection-observer* module abstracting standardized intersection observer API. When tracked component enters the viewport, a callback triggering an animation is executed. Example of more intricate animations are animated SVG illustrations in the introductory section and the first slide of the presentation component. Here the timing, type of motion and sequence of animation vary based on assigned class for which different rules apply.

5.3 Presentation

The *FactCheck Journey* section is a corner stone of the project. It uses a MUI stepper component as a wrapper for individual slides, walking to user through series of steps. These can be navigated either through buttons at the bottom or by clickable step numbers at the top of the component. Number of varying components were used in slides including, but not limited to, modal dialog, zoomable images, expandable accordion, pop-up helpers or interactive charts. The entry point for the `Presentation` component is located in the `process.tsx` page where all slides are imported and forwarded for display.

5.4 Gallery

Gallery, the subsequent section after the presentation showcases screenshots of existing applications, namely IdaFix and Dashboard. The gallery can be controlled either using buttons below the current image description or by swiping on the touch devices (as indicated by the hinting icon). Another component that display images is the Logo Slider component. As the name suggests, the component displays perpetually moving series of clickable logos that lead to respective website of given vendor or technology. At the same time, all the logos are desaturated by default. When hovering over a logo, the image gains color and is enlarged to indicate possible click action.

5.5 Demo

The Demo component marks the second most important section as it interactively demonstrates the simplified usage of the FactCheck framework. It is a combination of material `Tab` components, text editors and comparison result matrix (component displaying 4 numerical values). Overall it consists of 4 editable text editors with JSON-Ld content. Content of editors represent 4 entities, a local entity and 3 remote entities. The editors are implemented using Code Mirror library with addons to support JSON syntax highlighting and linting. In case of a syntax error, the erroneous location in the editor is highlighted with red markers. Additionally, affected tab is marked with an error icon and alert dialog listing affected editors is displayed. When contents of all editors contain valid JSON syntax, send button is enabled, allowing values to be sent for comparison. Clicking the send button triggers parsing of the editor contents into JSON data which is then sent via POST request to the server that processes incoming data and returns the comparison metrics. All 4 returned comparison metrics are subsequently displayed in the comparison matrix subcomponent.

5.6 Form

The last important component that been merely briefly touched on is the contact form. While MUI library provides abstractions of form input fields to ease the styling, state management is left to to the developer. The complexity of managed states in such form can grow fast as each field maintains at least 2 states specific for the field (validity and value) that can be modified by events (onchange and onblur). Depending on the state of input, the field can be either in error (red), valid (green) or unvisited state (primary color). Each input field requires custom validation. The form can be sent only after all fields are successfully validated. Successful form submit is concluded by success message in a dialog. In the current implementation, the server side validation is carried out by PHP script `emailHandler.php` located in the root of the public directory. Furthermore, the PHP script sends an email with the form contents to the specified destination. The user submitting the form will also receive an email with a copy of the submitted contents as a conformation.

6 Challenges

The most challenging part of the project was the configuration of it's constituents to facilitate mutual interoperability. For instance, configuration of the build process to enable static export required specification of custom image loader and base path used for routing. The initial plan was to use CSS-in-JS library Styled-Components (SC) in conjunction with MUI. However, this combination proved problematic as the style engine used in MUI conflicted with it's SC counterpart. This resulted in dropping of the SC dependency and focusing the styling efforts on the built in Styled JSX, offering similar functionality.

6.1 User testing

User testing has been carried out multiple times by total of 10 volunteers, both on mobile devices and desktop. It has uncovered numerous usability and comprehension issues. Some of those problems were:

- unintuitive presentation descriptions
- unexpected or missing helper pop ups
- jittery scrolling
- unresponsive Demo
- slow editor changes when typing in Demo
- performance bottlenecks during project filtration
- Confirmation message is not sent after form submission

All problems uncovered by the user testing have been located and fixed. Notably, the performance of Demo component has been significantly improved by using a debounced callback. This specific code construct delays the update of the component state that occurs in the editor by a specified amount of time. If a new debounced callback occurs before the timeout, the previous one is discarded. Since the content of the editor is stored both by the Code Mirror internal logic and the React component, fast pace changes don't need to be continually rewritten in the component's state on every key press. This approach results in considerable performance increases.

6.2 GitLab

Another challenging aspect was the automated deployment. Specifically, configuration of the GitLab pipelines initiating the build process and synchronizing the produced artifact with the remote server (as described in the Deployment section) proved error-prone. First, many of the GitLab functionalities (such as CI/CD or GitLab variables) are disabled by default and need to be enabled by administrator. Second, it is not sufficient to merely request a default version of target image. Proper version of node is crucial for successful build. Further, location of the build artifacts may vary depending on the GitLab server settings. Finally, default Ubuntu image does not contain any Linux utilities, namely ssh agent or rsync. Those need to be installed before an attempted use.

6.3 Cross-origin resource sharing

Cross origin resource sharing or COORS is a security mechanism implemented in all modern browsers that block resources with a different origin. In other words, resources can be requested but the returned response is blocked by browsers unless authorization header listing the requesting domain is present on the response. This security feature can't be disabled on the client and therefore requires modification of the API. The only affected component was the Demo, since it communicates with API on a different server. After the API was modified, the problem persisted. Despite accepting the JSON format as a response, the solution was to change the `"Accepted"` header format from `"application\json"` to `"application\xml"`.

Conclusion

The interactive presentation website built using NextJS and Material UI has fulfilled all defined requirements within the bounds of scheduled time frame. Built using the mobile-first approach, the website is responsive and optimized for performance. Usability issues uncovered during user testing have been fixed. The website interactively guides the user through essential concepts of the FactCheck framework, describes Architecture and show-cases existing projects. Additionally, user can filter the projects based on the status, send a message through the contact form and try out the simplified simulation of FactCheck entity comparison. The language of choice TypeScript with optional strong typing enabled static type checking, self-documenting of the components and easy extension of the code base. New projects and shortcomings that are rendered in components can be added into respective files. Build process has been configured to produce a static export that can be easily deployed on virtually any HTTP server. The project includes deployment configuration for GitLab pipelines that enables continuous delivery.