

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Designer for Interactive Exercises

Martin Hula

Supervisor: Ing. Karel Frajták, Ph.D.

Field of study: Open Informatics

Subfield: Software Engineering

January 2022

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hůla** Jméno: **Martin** Osobní číslo: **364623**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačů**
Studijní program: **Otevřená informatika**
Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Nástroj pro tvorbu interaktivních cvičení pro podporu výuky dějepisu pro ZŠ a SŠ

Název diplomové práce anglicky:

Designer for interactive exercises

Pokyny pro vypracování:

Cílem projektu bude návrh a implementace nástroje pro návrh, přípravu, a testování interaktivních cvičení. Cvičení jsou určena pro vzdělávací aplikaci pro práci s historickými prameny, kde s pomocí řady interaktivních nástrojů analyzují žáci v historické laboratoři krok po kroku dobové dokumenty, historické fotografie či zvukové záznamy. Příklady lze nalézt na <https://historylab.cz/>.

Nástroj budou používat didaktičtí pracovníci při přípravě nových cvičení. Cvičení je sekvence stránek s různým obsahem (text, fotografie, mapa, audio záznam) a úkoly pro žáky. Didaktici budou moci cvičení sestavit z jednotlivých nástrojů, nahrávat historické prameny (fotografie, mapy, audio/video záznamy) a také si cvičení vyzkoušet než bude uvolněno pro studenty.

Nástroj musí umět pracovat s již existujícími cvičeními, proto je nutné cvičení analyzovat a na základě této analýzy definovat nejvhodnější reprezentaci pro jednotlivé nástroje. Do budoucna je nutné počítat s přidáváním dalších nástrojů.

Celý systém by měl být řádně otestován.

Seznam doporučené literatury:

<https://historylab.cz/>
<https://angular.io/resources/?category=development>
<https://material.angular.io/>

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Karel Frajták, Ph.D., laboratoř inteligentního testování systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **25.05.2021**

Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **19.02.2023**

Ing. Karel Frajták, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

I want to express my gratitude to my thesis supervisor Ing. Karel Frajták, Ph.D, for his guidance and support during my work on this thesis. He was always available to answer my questions and helped me understand the HistoryLab.cz project.

I also need to thank my family for their constant support and understanding, especially when I could not spend as much time with them as I would have liked.

Declaration

I hereby declare that I am the sole author of this thesis. It is my original work. I have cited all my sources of information, all in accordance with the methodological guideline on upholding ethical principles during the preparation of the final academic theses.

In Prague on 4. January 2022

Abstract

Project HistoryLab.cz provides teachers with a convenient educational platform, which can be used to give interactive exercises to students during their class. However, there is no easy way to create these exercises at present. Creating a new exercise involves collaboration between the didactic staff, which defines the exercise contents, and HistoryLab's developers, who have to code the new exercise. This thesis aims to create an Exercise Designer application that can simplify the process of exercise creation and give more power to the didactic staff. With such an application, didacticians should be able to create draft exercises on their own, reducing the time needed to communicate with the developers. The first part of this thesis describes an analysis of the requirements and the process of deciding Exercise Designers' key architectural elements. The analysis has been done with a focus on future extensibility and support for adding more tools in the future. The second part describes the implementation of the Exercise Designer as a web application. Although it does not yet support all of HistoryLab's tools, its extensibility should make it easy to add these in the future. User testing has been conducted, and improvements to the application have been made based on the test results.

Keywords: exercise, designer, historylab, json, schema

Supervisor: Ing. Karel Frajták, Ph.D.
FEE CTU, Karlovo náměstí 13, Praha 2

Abstrakt

Projekt HistoryLab.cz dává učitelům dispozici výukovou platformu, která může být použita pro zadávání interaktivních cvičení studentům. V současné době však neexistuje jednoduchý způsob, jak tato cvičení vytvářet. Tvorba každého cvičení vyžaduje spolupráci didaktiků, kteří definují obsah cvičení a vývojářů HistoryLabu, kteří obsah převedou do formátu čitelného aplikací. Cílem této práce je vytvořit aplikaci návrháře cvičení, která zjednoduší proces tvorby cvičení. Sami didaktici budou schopni vytvořit a vyzkoušet své vlastní cvičení, a tak se sníží nutnost jejich komunikace s vývojáři. První část této práce popisuje analýzu požadavků a proces rozhodování architektury klíčových částí aplikace. Analýza bere v úvahu rozšiřitelnost aplikace a přidávání nástrojů v budoucnosti. Druhá část této práce popisuje implementaci aplikace návrháře cvičení jako webové aplikace. I když v současné době tato aplikace ještě nepodporuje všechny nástroje HistoryLabu, díky rozšiřitelnosti by nemělo být obtížné je přidat. Aplikace prošla uživatelským testováním a na základě výsledků byla upravena.

Klíčová slova: cvičení, návrhář, historylab, json, schema

Překlad názvu: Nástroj pro tvorbu interaktivních cvičení pro podporu výuky dějepisu pro ZŠ a SŠ

Contents

| | | | |
|--|-----------|--|-----------|
| 1 Introduction | 1 | 2.3 Technical Requirements | 12 |
| 1.1 Background | 1 | 3 Task Analysis | 15 |
| 1.1.1 HistoryLab | 1 | 3.1 Exercise Schema Analysis | 15 |
| 1.1.2 HistoryLab Exercise | 2 | 3.2 First Implementation Ideas | 16 |
| 1.2 Actors | 3 | 3.2.1 Exercise JSON Printer Prototype | 16 |
| 1.3 Motivation | 4 | 3.2.2 Recursive Form Prototype | 16 |
| 1.3.1 Exercise Creation Workflow | 4 | 3.3 JSON Forms Analysis | 17 |
| 1.3.2 Communication | 6 | 3.3.1 Problems of JSON Forms | 17 |
| 1.3.3 Communication Problem Statement | 8 | 3.3.2 Possible Solutions | 19 |
| 1.3.4 Potential Solutions | 8 | 3.3.3 Conclusion | 22 |
| 1.4 Thesis Structure | 9 | 3.4 Schema-driven Form Prototype | 23 |
| 2 Fundamentals | 11 | 3.5 Improvements to the Schema | 23 |
| 2.1 Problem Statement | 11 | 3.6 UI Form Generation | 24 |
| 2.2 Requirement Analysis | 12 | 3.7 Extensibility Analysis | 26 |
| 2.2.1 Functional Requirements | 12 | 3.7.1 Extensibility with JSON Schema | 26 |
| 2.2.2 Non-functional Requirements | 12 | 3.7.2 Extensibility of the Code | 26 |

| | | | |
|-----------------------------------|-----------|---|-----------|
| 4 Design | 29 | 6 Testing | 43 |
| 4.1 Application Architecture..... | 29 | 6.1 Static Analysis | 43 |
| 4.1.1 Front End..... | 30 | 6.2 Unit Tests | 44 |
| 4.1.2 Back End | 30 | 6.3 API Testing | 44 |
| 4.1.3 Database | 31 | 6.4 User Testing | 44 |
| 4.2 Exercise Tools | 31 | 6.4.1 Methodology | 44 |
| 4.3 Design Prototype | 32 | 6.4.2 Tasks..... | 45 |
| 5 Implementation | 35 | 6.4.3 Results | 46 |
| 5.1 Overview | 35 | 6.4.4 Changes in Response to Users' Feedback | 48 |
| 5.1.1 Front End..... | 35 | 7 Conclusion | 51 |
| 5.1.2 Back End | 37 | Limitations | 51 |
| 5.1.3 Database | 37 | Summary | 51 |
| 5.1.4 Interactions | 37 | Future Improvements..... | 52 |
| 5.2 Key Technology | 38 | Bibliography | 53 |
| 5.3 Exercise Tools | 40 | A DVD | 55 |
| 5.4 Implementation Details..... | 40 | B User Testing Results | 57 |
| 5.4.1 Schema Conversion..... | 40 | C Images | 63 |
| 5.4.2 Schema Representation | 41 | | |

Figures

| | | | |
|--|----|--|----|
| 1.1 HistoryLab Exercise Catalogue Example | 2 | C.1 Example Exercise - Slide One . . | 63 |
| 1.2 Example Sequence of Exercise Slides (see Appendix C) | 3 | C.2 Example Exercise - Slide Two . . | 64 |
| 1.3 Exercise JSON File Example | 5 | C.3 Example Exercise - Slide Three | 64 |
| 1.4 Exercise Creation Flowchart | 6 | C.4 Example Exercise - Slide Four . . | 65 |
| 1.5 Trello Task Example | 8 | C.5 Example Exercise - Slide Five . . | 65 |
| 3.1 Example Data and Form Generated by JSON Forms | 17 | C.6 Exercise Designer - Profile Editor (Example Slide) | 66 |
| 3.2 Example Schemas used by JSON Forms | 18 | C.7 Slide Editor | 67 |
| 4.1 Proposed Database Schema | 31 | | |
| 4.2 Exercise Structure Illustration . . | 32 | | |
| 4.3 Exercise Designer Visual Prototype | 33 | | |
| 5.1 Profile Editor | 36 | | |
| 5.2 Slide Editor | 36 | | |
| 5.3 Sample Interactions | 38 | | |

Tables



Chapter 1

Introduction



1.1 Background



1.1.1 HistoryLab

Project HistoryLab.cz has started in 2016 as an initiative to aid teachers of middle schools and high schools by creating an interactive web application, which the students can use during a history class. „HistoryLab is an educational application that works with historical sources” (“HistoryLab”, 2021), with a particular focus on the historical topics of the 20th century. It is composed of a set of unique tools for teaching history in an innovative way, and it provides teachers with a catalogue of exercises that they can use to teach their students (example shown in Figure 1.1).

The project has been created thanks to the grant from the Technology Agency of the Czech Republic¹, and the main participants in this project are the Institute for the Study of Totalitarian Regimes (USTR)², Faculty of Electrical Engineering of the Czech Technical University in Prague³, Masaryk Institute and Archives of the CAS⁴, Fraus Publishing⁵, Faculty of Education

¹<https://www.tacr.cz/en/>

²<https://www.ustrcr.cz/en/>

³<https://fel.cvut.cz/en/>

⁴<https://www.mua.cas.cz/en>

⁵<https://www.fraus.com/>

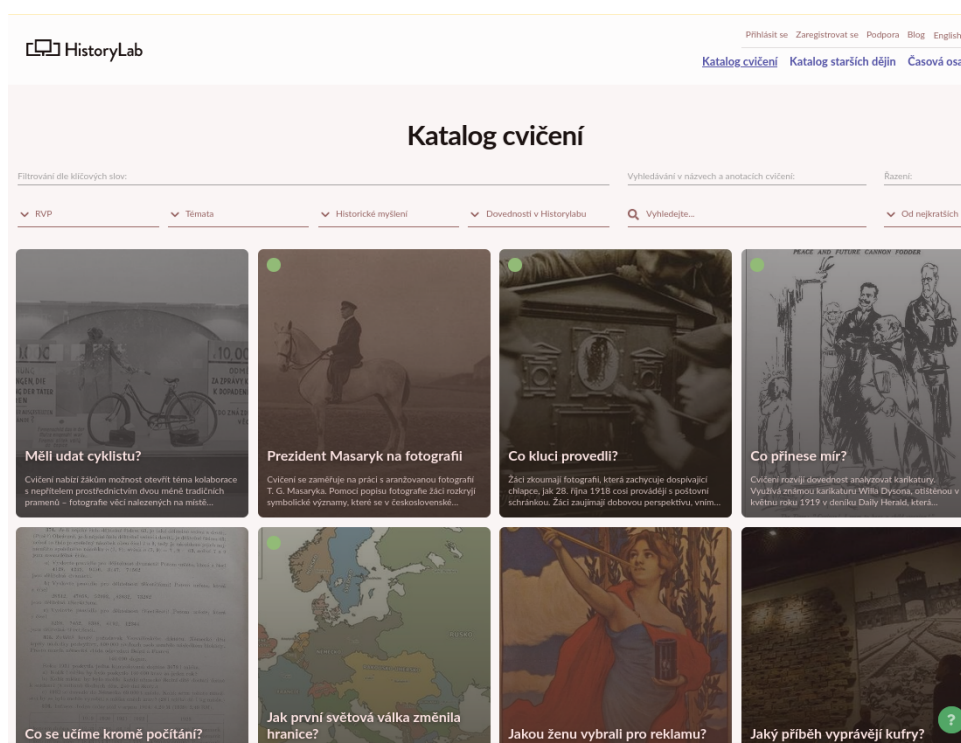


Figure 1.1: HistoryLab Exercise Catalogue Example

of Charles University⁶ and the Institute of Contemporary History (USD)⁷. (“Historylab: Using Technology... - cep - ta CR starfos”, 2018) The development started in 2016, and in the first two and half years of development, a prototype of the HistoryLab application was created. Dozens of teachers tested it while gathering feedback about the educational process. Currently, HistoryLab offers its own learning management system, where teachers can assign individual exercises to students and provide feedback. It had become a pilot program to improve the way history classes have been taught, and it was a great solution during the period of the Covid-19 pandemic when some of the lessons were conducted online.

1.1.2 HistoryLab Exercise

Exercise in the HistoryLab application is a sequence of slides of various types, which are composed of multiple interactive tools. Each tool offers a different way of interaction with the content. Students can use these tools to analyse

⁶<https://pedf.cuni.cz/PEDFEN-1.html>

⁷<https://www.usd.cas.cz/en/>

historical documents, photographs or audio recordings, and other types of content.

■ Example

Let us review an existing exercise called „What they expected of the sewing machine?“ (see Figure 1.2). This exercise is composed of five slides. In the first slide, students are presented with a photo, which they should study closely. In the second slide, the students are expected to mark essential parts of the photo and add a short description. The third slide lets the students select several keywords closely related to the previous photo. The fourth slide is composed of a gallery of photos on the left side and a paragraph of text. Students are expected to highlight relevant text parts, using different colours for different concepts. The last slide has a gallery of photos on the left side and empty text fields on the right side. Students are expected to provide their answers to questions in the text fields.

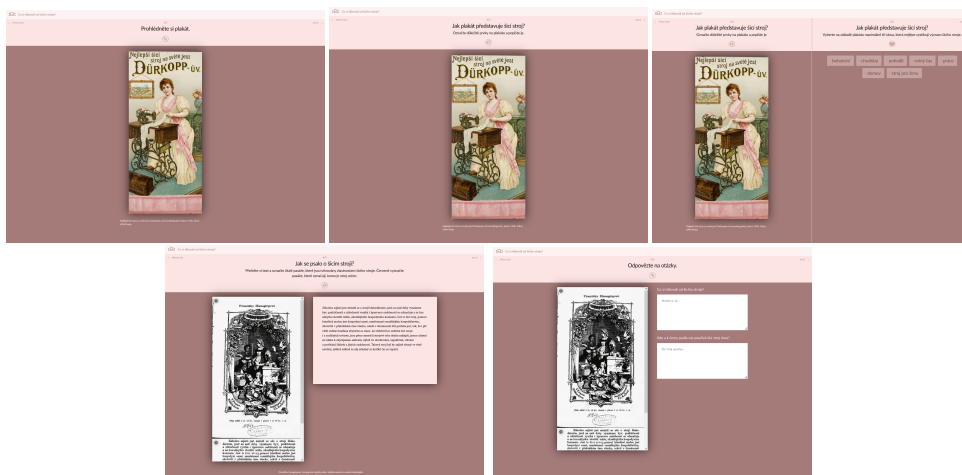


Figure 1.2: Example Sequence of Exercise Slides (see Appendix C)

■ 1.2 Actors

- **Didactic staff.** These actors are exercise creators. They analyse the historical sources and are responsible for bringing their exercise design ideas to life.
- **Exercise developers.** These actors are programmers, coding exercises into the HistoryLab application’s format and releasing them for public use.

- **Teachers.** Teachers use the published exercises in class and assign them to their students.
- **Students.** Students are the final end-users of the exercises, interacting with the application and working on the tasks.

■ 1.3 Motivation

■ 1.3.1 Exercise Creation Workflow

1. First Ideas. Didactic staff finds a historical concept that is adaptable to the format of the HistoryLab exercise.

2. Exercise Script. Didactic staff transforms their idea into a concrete (presentation) slides flow. In particular, the slides' size and functions need to correspond with the capabilities of the HistoryLab application. Didactic staff is familiar with the application, the available tools and processes.

3. Coding. Exercise developers use the exercise script to create the final exercise used in the HistoryLab application. Technically, this exercise is represented mainly as a set of media files and the exercise definition JSON file (example shown in Figure 1.3).

4. Testing. Didactic staff tests the final exercise in the HistoryLab application. This is important in order to see if the exercise is both formally and substantively correct and if it matches the original expectations.

There are three possible results of this testing stage:

1. The exercise is deemed acceptable and is approved for public release.
2. The exercise has some minor flaws which can be corrected. In this case, didactic staff needs to give feedback to the developer, and the task goes back to the coding stage, where the exercise is modified as necessary.
3. The exercise has some major flaws which cannot be corrected. In particular, the exercise may not match the didactician's expectations, or there are some other significant obstacles why this exercise cannot be

```

1 {↵
2   "cviceni": {↵
3     "version": {↵
4       "global": "<%= version %>",↵
5       "content": 1↵
6     },↵
7     "katalog": [↵
8       "test"↵
9     ],↵
10    "language": "cs",↵
11    "slug": "co-si-slibovali-od-siciho-stroje",↵
12    "nazev": "Co si slibovali od šicího stroje?",↵
13    "autor": [↵
14      "Václav Sixta"↵
15    ],↵
16    "klicovaSlova": {↵
17      "rvp": [↵
18        "technika"↵
19      ],↵
20      "koncept": [↵
21        "gender"↵
22      ],↵
23      "b4": [↵
24        "trvání a změna"↵
25      ],↵
26      "historylab": [↵
27        "formulujeme a ověřujeme hypotézu"↵
28      ],↵
29    },↵
30    "casovaOsa": {↵
31      "roky": [↵
32        1900↵
33      ],↵
34      "epochy": [↵
35        {↵
36          "refId": "czech",↵
37          "obdobi": [↵
38            "habsburská monarchie a její rozpad"↵
39          ],↵
40        }

```

Figure 1.3: Exercise JSON File Example

corrected. In this case, the didactic staff must completely rework her script, moving the task back to the second stage. Or she can even decide to scrap the exercise altogether.

5. Release. Exercise developers release the exercise for public use.

As an example. A didactician will come up with an interesting historical concept of the introduction of the sewing machine. She will gather some related photographs, texts and come up with a list of interesting questions about sewing machines. Then she creates a sequence of slides that should

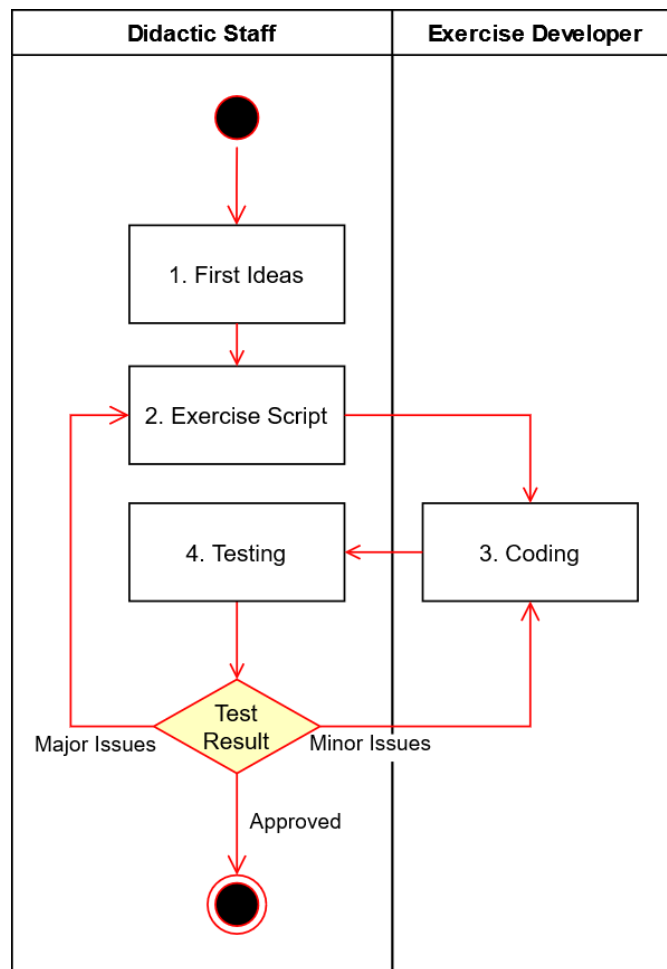


Figure 1.4: Exercise Creation Flowchart

be compatible with the exercise format and then send it to the developer for coding. She cannot try it out if this exercise works as expected. At some later point, the developer creates the exercise. The didactician then tests the exercise but finds out that some of the texts need to be updated, and also the order of images would better be changed. She gives her feedback to the developer, who, at some later point, will incorporate these changes into the exercise. This cycle continues until the exercise is ready.

■ 1.3.2 Communication

In the exercise creation process, two actors need to participate in communication in order to create an exercise. Didactic staff needs to work closely with the exercise developers between stages two to four (from Exercise script to Testing). In each of these steps, there are various potential reasons for this

communication need.

■ Common Communication Patterns

Firstly, between stages 2 and 3, the didactic staff needs to present their exercise script to the exercise developer. At this point, the developer may have some technical feedback for the didactician.

Secondly, during stage 3, the developer may have some additional questions for the didactician if the exercise script was not clear enough or there was some other technical difficulty during the development.

Thirdly, after testing in stage 4, the didactic staff needs to give her feedback to the exercise developer. After discussion between the didactician and the developer, the final testing feedback is decided, falling into one of the three possible results.

If the result of testing were not satisfactory, the task would go back to stage two or three, and the communication patterns mentioned above will occur all over again.

■ Communication Tool

In the current workflow, the exercise creation process is managed as a Trello project⁸. For each new exercise, the didactic staff creates a task on the Trello board, assigned to the exercise developer (example shown in Figure 1.5). This task includes the exercise script and all the required audio and visual media files. All communication between the didactic staff and the developer is done in the comments attached to this task. As such, this communication is inherently asynchronous. Both sides have to wait for a non-predetermined amount of time for the other side's reply. This makes the communication very inefficient as it requires a lot of unnecessary mental context switching. Even simple adjustments to the exercise may take a long time to be resolved.

⁸<https://trello.com/about>

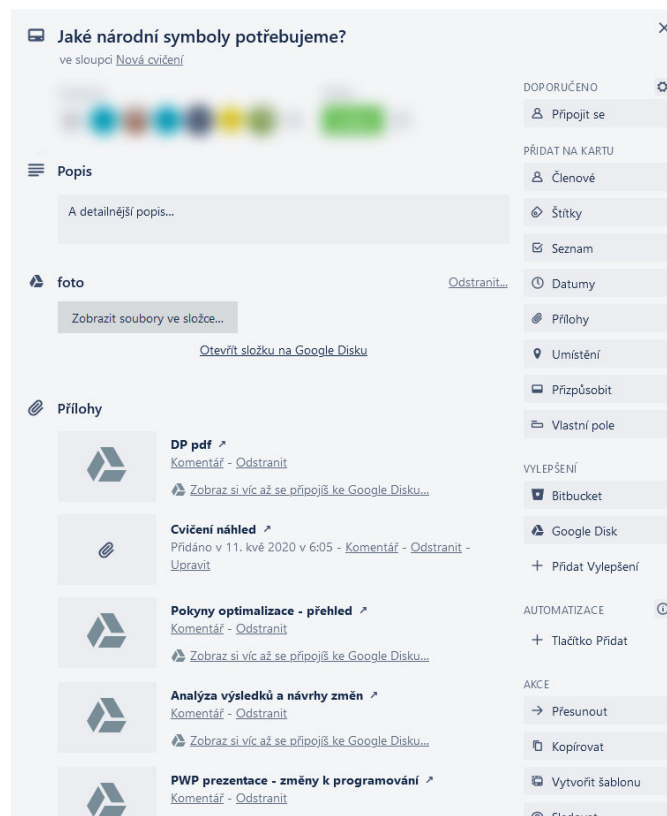


Figure 1.5: Trello Task Example

1.3.3 Communication Problem Statement

Communication between the didactic staff and the exercise developers is inefficient and creates an unnecessary overhead.

1.3.4 Potential Solutions

1. Make the existing communication more efficient. Changing the tools in use may help streamline the communication. Using real-time chat or video calls may help resolve many individual issues simultaneously.
2. Create an exercise designer tool to allow the didactic staff to create exercises themselves. Instead of making communication more efficient, this would reduce the amount of communication. In the current workflow, the developer's presence is required in order to make any changes to the exercise. Creating an exercise designer tool would allow the didactic staff to create their own exercise prototypes. They could see if their idea

works without the need to communicate with the developers.

■ 1.4 Thesis Structure

Inspired by the list of key circumstantial elements of rhetorics created by Boethius: „Quis, quid, cur, quomodo, ubi, quando, quibus auxiliis” (Robertson, 1946), this thesis is structured to answer the important questions of „Who, what, why, how, where, when, with what”.

Thesis Assignment. The assignment page introduces "who" is the author of the thesis, the institution "where" this thesis had been written, as well as the time "when" this thesis had been written.

1. Introduction. The thesis introduction describes the background of the project. It introduces the stakeholders "who" are interested in this project and explains "why" this project exists and its motivation.

2. Fundamentals. This chapter introduces the Exercise Designer project and describes "what" are the project's goals and "with what" technologies it should be created.

3. Task Analysis. This chapter explains "how" key parts of the Exercise Designer were created and why. It explains the thought process of analysing the individual requirements and decisions on essential architectural elements.

4. Design. This chapter presents the details about "how" the project was designed based on the requirements and results of the analysis.

5. Implementation. This chapter explains "how" the project was implemented and what are the key architectural elements.

6. Testing. This chapter shows "how" the project was implemented quality-wise. It describes the process of user testing and improvements made as a result of this testing.

7. Conclusion. This chapter presents the summary of the project. It concludes with regards to "what" has been done and suggests possible improvements that could be done in the future.



Chapter 2

Fundamentals



2.1 Problem Statement

This project focuses on solving the problem described in the motivation. This project aims to create an interactive application, Exercise Designer, which will allow the didactic staff to design, prepare, and test new exercises on their own. They could create draft exercises, confirm their expectations and see which ideas work well in the exercise format and which do not. In the end, these draft exercises may or may not be used to produce the final version of the exercise.

It is necessary to analyse the existing HistoryLab application and its exercises definition. Each exercise uses a set of tools. Exercise Designer needs to create the best possible representation for these tools to be able to use them during the creation of new exercises.

■ 2.2 Requirement Analysis

■ 2.2.1 Functional Requirements

1. User should be able to create new exercises and modify the existing ones.
2. User should be provided with a variety of tools that can be used to build an exercise.
3. User should be able to upload files representing historical sources.
4. User should be able to test the exercise before it is released to students.
5. Exercise Designer should be able to read the existing exercises and create exercises compatible with the HistoryLab application.

■ 2.2.2 Non-functional Requirements

1. Exercise Designer should be extensible. More tools will be added in the future.
2. Exercise Designer should be properly tested.
3. Exercise Designer should be easy to use.

■ 2.3 Technical Requirements

■ Web Application

Exercise Designer should be implemented as a web application written in Javascript. HistoryLab application is also a web application, and the didactic staff will use both HistoryLab as well as this new Exercise Designer. Using the same technology allows for a seamless transition between these two applications. Another advantage is that web application does not need to be installed locally on the user's computer. This gives the flexibility of using the application from anywhere.

■ React

Exercise Designer should use the React Framework. React was the most used front end framework in 2020 (Greif and Benitte, 2021). Other developers in the HistoryLab team are also familiar with it and can contribute to this project in the future. Using Javascript allows for smooth interaction with the JSON (Javascript Object Notation) format of exercises and JSON Schema validation libraries.

■ Material Design

Exercise Designer uses Material Design in its views in order to create a consistent user experience without "reinventing the wheel" (custom development of each visual component is not the purpose of this work). Material UI library (React implementation of the Material Design)¹ provides many of the essential components and layout capabilities, support for themes, icons and many other features.

■ JSON Schema

Exercise Designer needs to understand and support the existing exercises and create new exercises that are compatible with the exercise application. Since exercises are written in the JSON format, JSON Schema² is used for the purpose of establishing and maintaining the exercise JSON file validity.

JSON Schema specification document is a JSON file describing the schema of a JSON data file, which allows verifying the structure and data types within that data file and contains additional metadata about this file.

¹<https://mui.com/>

²<https://json-schema.org/>

■ JSON Forms

Exercise Designer should use JSON Forms³ library in order to generate form fields based on the existing JSON Schema. JSON Forms accepts a schema and the related data as an input and dynamically generates an editable form (see Figure 3.2). This will allow for a fully extensible representation of the editor.

³<https://jsonforms.io/>



Chapter 3

Task Analysis



3.1 Exercise Schema Analysis

One of the first tasks of this thesis was to analyse the existing exercises in order to be able to create a generic representation for the individual tools within the exercises but also to make sure that the existing exercise schema has no structural problems or inconsistencies, which would make this task more difficult.

The first step was to manually create a text-based representation of the existing exercise JSON files. Going through the individual exercise files manually allowed me to get very familiar with the exercise structure and gain an insight into any potential structural issues and improvements, which is something I simply would not have, were I to leverage a fully automated solution.

The second step was to create a JSON Schema representation of the exercise JSON files. A representative sample of 8 exercise files was chosen from the complete set of 150 exercises in order to simplify the schema creation process. This representative sample contained all of the tools used throughout the complete set of exercises and, as such, should allow for a creation of a JSON Schema applicable to the complete set of exercises.

■ 3.2 First Implementation Ideas

Initial ideas on the implementation of the Exercise Designer were based on the requirement of extensibility and the fact that the exercises are represented as JSON files. The goal was to base the designer's behaviour purely on the exercise JSON data file and its respective JSON Schema file. Thus came the idea of having a dynamically generated form as the designer's core. Such a form would be fully extensible without the need to change the application source code. Depending purely on the Exercise JSON Schema would mean that adding more tools in the future would simply mean updating a single file, and the application's behaviour would adapt accordingly.

■ 3.2.1 Exercise JSON Printer Prototype

This first simple prototype was used to analyse the difficulties of displaying the contents of an exercise JSON file. In particular, the focus was on handling the iteration over different types of slides within the exercise. Every type of slide contains various tools and as such, correctly identifying the type of slide is very important.

■ 3.2.2 Recursive Form Prototype

This prototype was used to explore the possibility of dynamically generating a form based on exercise JSON. Using exercise JSON as an input, this prototype uses recursion over the tree-like data structure in order to dynamically generate React form components with different levels of nesting. Each form field is editable, and changes are reflected in the JSON data.

This prototype proved that generating a form based on exercise data is viable. But to fully cover the requirements, it was necessary to incorporate JSON Schema and the related metadata in the form generation.

3.3 JSON Forms Analysis

Initial ideas on incorporating JSON Schema-based form generation were based on using a 3rd party library called JSON Forms¹. This library allows for dynamic generation of a form based on JSON Schema with little additional development overhead, which would simplify my development. However, two problems appeared after a thorough investigation of the library's features and restrictions.

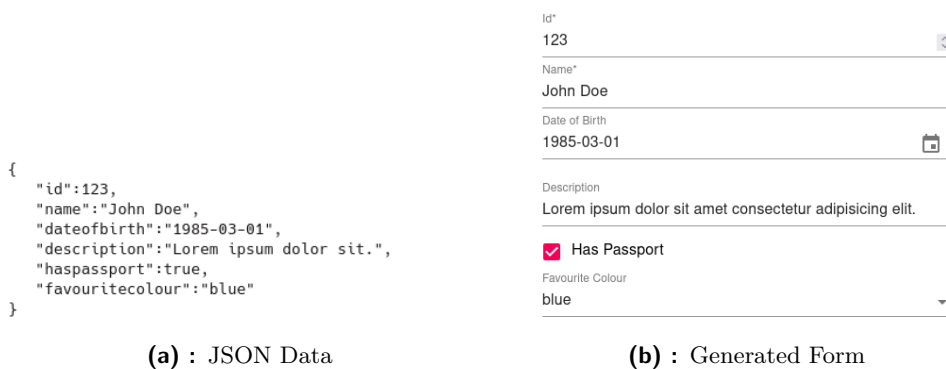


Figure 3.1: Example Data and Form Generated by JSON Forms

3.3.1 Problems of JSON Forms

1. JSON Forms library does not support the complete set of attributes in JSON Schema. As such, it is not possible to use this library with the previously built representative JSON Schema for exercises.

In particular, these were the restrictions presented by the library:

- Limitations to nested schema structure, where nested schemas were not considered required even when set to be so.
- No support for a conditional extension of a schema object on the same level of nesting (something that was very much present in the current JSON exercise structure).
- No support for `const` keywords or `if-then-else` syntax.
- No support for extending an object based on an external reference.

¹<https://jsonforms.io/>

2. JSON Forms library needs an additional UI Schema file to define the visual layout of the generated form (see Figure 3.2). This library can produce only a very basic form layout without this additional schema. Many of the features that should be supported from the JSON Schema alone are only supported when this additional file is present.

In particular, without an additional UI Schema:

- There can be undesired or limited behaviour in certain field types. For example, inputs based on enum values allow empty input even when an empty value is not in the list. (Hall, 2019).
- It is not possible to generate a form that supports conditional behaviour. For example, it is not possible to generate different form fields based on the selected value of an enum field.

| | |
|--|---|
| <pre> { "type": "object", "properties": { "id": { "type": "integer", "default": 0 }, "name": { "type": "string" }, "dateofbirth": { "type": "string", "format": "date", "default": "2000-01-01" }, "description": { "type": "string", "default": "default content" }, "haspassport": { "type": "boolean", "default": false }, "favouritecolour": { "type": "string", "enum": ["red", "green", "blue"] } }, "required": ["id", "name"] } </pre> | <pre> { "type": "VerticalLayout", "elements": [{ "type": "Control", "scope": "#/properties/id", "label": "Id" }, { "type": "Control", "scope": "#/properties/name", "label": "Name" }, { "type": "Control", "scope": "#/properties/dateofbirth", "label": "Date of Birth" }, { "type": "Control", "scope": "#/properties/description", "label": "Description", "options": { "multi": true } }, { "type": "Control", "scope": "#/properties/haspassport", "label": "Has Passport" }, { "type": "Control", "scope": "#/properties/favouritecolour", "label": "Favourite Colour" }] } </pre> |
| (a) : JSON Schema | (b) : UI Schema |

Figure 3.2: Example Schemas used by JSON Forms

3.3.2 Possible Solutions

1. Not to use JSON Forms library, create a static form. Instead of a dynamically generated form based on JSON Schema, it is possible to create a static form based on the current knowledge of the Exercise JSON structure. This would mean not using JSON Schema for the form generation, but it could still be used for validation. This type of form could still be extensible by designing the underlying code and component architecture in an extensible way.

Advantages:

- Flexibility in design. Not generating the content would mean that the individual components can be fully customized to their particular use case. The developer can directly define business rules directed at the behaviour and relationships between the components. In other words, the resulting user experience could be much less of a form and much closer to the exercise layouts and designs presented by the HistoryLab application, with which users are already familiar.

Disadvantages:

- Reduced extensibility. Using a standardized format like JSON Schema to generate the form ensures extensibility where a developer is forced to adhere to this standard. Not depending on JSON Schema could mean less extensibility. The developer's interpretation of extensibility may vary, and the final product may be less extensible as a result.

2. Restructure exercise JSON Schema. Require an additional UI Schema. Since the JSON Forms library does not support the current set of attributes needed by the exercise JSON Schema, it should be possible to rewrite the schema so that only the supported attributes are used.

Advantages:

- Form autogeneration. This change would enable the use of the JSON Forms Library, as initially expected. The form would be autogenerated based on the JSON Schema file. Having an additional UI Schema would mean that the layout could be customized as needed.

Disadvantages:

- Loss of validation accuracy. Changing JSON Schema only to use a supported subset of attributes may mean falling back to a more generalized version of the schema. It may not be possible to express all the requirements on schema validity fully. For example, instead of validating a subset of the required fields based on some condition, this condition may need to be removed, and these fields may all need to be set as optional.
- Duplication in JSON Schema. No support for extending an object based on external reference would mean that it would be necessary to duplicate the definitions instead of simply referring to another schema.
- Loss of generality in the schema definition. By only allowing a subset of JSON Schema supported by the JSON Forms library to be used in the exercise schema definition, there is a loss of generality in the schema definition. The exercise JSON Schema would not be defined based on the precisely defined set of rules of the JSON Schema standard but instead by some vaguely defined subset which this library supports.

3. Restructure exercise JSON data files and create a new JSON Schema.

Require an additional UI Schema. The most significant issue with the JSON Forms library is caused by conditional structuring of the current exercise JSON files (conditional extensions of schema and sub-schema on the same level of nesting, which could be refactored by splitting into multiple objects or encapsulating these changes as children of a new parameter). This issue can be avoided by restructuring the exercise JSON files to express the same behaviour more simply.

Advantages:

- Cleaner exercise JSON structure. The current same-level conditional extensions are difficult to understand and unnecessary.
- Simpler and easier-to-understand schema. Cleaning the JSON structure would result in a simpler schema with fewer conditionals.

Disadvantages:

- Scope too large. Refactoring the structure of exercise JSON files would require updating all of the existing exercise JSON files and how the HistoryLab application uses these files.

- Does not solve all problems with JSON Forms. While this may solve the issue of validation accuracy, other disadvantages from the previous point are still present.

4. Convert JSON Schema to a schema supported by JSON Forms. Create a conversion mechanism that would transform a valid exercise JSON Schema to a different schema, which could be used by JSON Forms. This conversion could also produce the relevant UI Schema.

Advantages:

- Accurate JSON Schema validation. Schema validation can be still be done on the original JSON Schema.
- Form autogeneration. JSON Forms can use the converted schema to autogenerate the form.
- Can generate UI Schema. Schema conversion allows generating another UI Schema file, which can customize the visual layout of the form.

Disadvantages:

- Complex implementation. Converting a JSON Schema into a different schema would require a code that can read the input schema, understand the mapping between generic JSON Schema and a subset of attributes valid in JSON Forms, and convert between these schemas.

5. Not to use JSON Forms library, implement a custom form generation based on the JSON Schema. If it is too difficult to integrate the JSON Forms library, another option is to implement the feature from scratch. Implementing a custom code that can dynamically generate a form based on the JSON Schema is possible.

Advantages:

- Accurate JSON Schema validation. Can use original JSON Schema — custom implementation is not restricted to any subset of attributes.
- Form autogeneration. The form will be autogenerated based on the JSON Schema.
- Does not use JSON Forms. There is no need to adapt to specific requirements of this library, and as such, the code is cleaner and simpler.

Disadvantages:

- Complex implementation. Similar to the previous point, it is necessary to create code that can understand the JSON Schema and generate form components as a result.

6. Use a different library. JSON Forms is not the only library that can be used to generate a form based on JSON Schema dynamically.

Advantages:

- Can solve problems of JSON Forms

Disadvantages:

- May bring different problems, which will require further analysis.
- No better alternative. As per my research, JSON Forms is the most comprehensive library for this purpose. The best possible alternative - react-jsonschema-form - seems to have an even worse support of JSON Schema attributes and does not support external references.

■ **3.3.3 Conclusion**

While solution one would offer the best flexibility, it would limit the extensibility of using JSON Schema.

Solutions two to four would put a lot of effort to adapt the code to the requirements of the JSON Forms library, leading to compromises in the design and potentially a lot of unnecessary structural overhead. Furthermore, depending on the library may restrict future development.

I have decided that solution five provides the most benefits in the current situation. While more complex implementation-wise, implementing a custom form generation will allow for flexibility in adapting the resulting form to whatever the Exercise Designer requirements may be.

I have also recognized that the current exercise JSON Schema is not optimal and has some problems that should be addressed. I will create a modified,

cleaner version of the schema. In order to avoid having to refactor all of the existing exercise JSON files and the exercise application itself, I will create a conversion tool capable of bidirectional translation between the current and the newly created version of the JSON Schema.

3.4 Schema-driven Form Prototype

The problem of the Recursive Form prototype was that it would generate form purely based on the exercise data JSON file. Optional properties not present in the data would not be displayed. Furthermore, the form would not have access to the exercise metadata and would be further limited.

Schema-driven Form builds on the original idea of Recursive Form and expands it. This prototype uses both exercise data JSON file as well as JSON Schema representation as an input. As such, it can generate the entire form based on the JSON Schema, having access to all the form's metadata, and it can fill this form with values present in the exercise data JSON file.

3.5 Improvements to the Schema

As a result of the previous analysis, I have concluded that the existing structure of exercise JSON files is not without problems. Firstly, there are problems with building a fully functional JSON Schema representation (there are limits to what can be expressed in JSON Schema, and only a subset of these expressions is supported by libraries that leverage JSON Schema). Secondly, there are problems regarding the basic structure and ease of understanding of these JSON files.

Renamed Properties

The simplest cleaning of the schema was in renaming some of the properties to better reflect the semantic of their use:

- `cviceni` renamed to `profil`
- `duplikovat` renamed to `reference`

- `svg` renamed to `dragAndDrop` if used for drag and drop, `pretahovani` and `soubory` renamed to `dragItems` and `dropItems`
- `klicovaSlova.klicovaSlova` renamed to `klicovaSlova.skupiny`

■ Enforced Data Type Consistency

Some properties in the schema would accept different data types for no apparent good reason:

- `class` property must be an array of strings. It was either a string or an array of strings before.
- `napoveda.text` property must be a string. An array of single string was allowed before.
- `uzivatelskyText.layout` moved to `uzivatelskyText.nastaveni.layout`, to be consistent with all the other types of slides.

■ Other Structural Improvements

- Redundant `drop` boolean property in `dropItems` removed.
- Class `half-slide-previous` was dropped and `half-slide-active` replaced by a single boolean property `rozpulitSPredchozimSlajdem` (split with the previous slide).
- Created a root property `media`, grouping all media files with their unique ids. All media references with `soubor url` changed to `ref id` references to `media`.
- `profil.casovaOsa.epochy` changed from a tuple array of two objects to a single object.

■ 3.6 UI Form Generation

Schema-driven form prototype proved that it is possible to have a dynamically generated form based purely on JSON Schema representation of the exercise. This allows for the highest level of extensibility by simply modifying the

schema. However, using this approach to implement the Exercise Designer has its limitations.

■ Limitations

- Limited customization. It would be challenging to customize individual fields based on their semantics or meaning to the user. Semantic customizations would undermine the generality of the implementation.
- Base element would be a form. All basic structural elements of the Exercise Designer would be generated form fields.
- Poor user experience. It would restrict the type of interactions a user can have with the system, and the overall user experience would be poor.

■ Potential Solution

Combine statically-defined and dynamically generated parts. If it were possible to ease the restriction on modifiability only through the JSON Schema, it would be possible to design the system as a combination of dynamically generated parts based on the JSON Schema and static parts (still extensible with code modifications), which could allow for semantic differentiation and additional ways of interacting with the system, all this leading to better user experience.

■ Conclusion

This is simply a problem of trading extensibility for user experience. Would the improvements in user experience justify the decrease in extensibility?

From my understanding, the goal of this project is to make it easier for users to create exercises. Having a fully generated form representing the JSON would not bring a lot of benefit to the end users over editing the JSON file directly. A combined approach should offer a much better overall user experience while still keeping a level of extensibility and responding to changes in the JSON schema.

■ 3.7 Extensibility Analysis

Exercise Designer should be extensible. Because there can be many different interpretations of extensibility (CWN, 2016), let us define what it should mean in the context of this application.

The application should be extensible with regards to adding more tools in the future, and it should, to some extent, react to changes to the associated JSON Schema representation.

■ 3.7.1 Extensibility with JSON Schema

As discussed before, Exercise Designer will not be completely extensible using only JSON Schema. This would be too restrictive. However, the exercise schema can be split into two essential parts - exercise profile and exercise slides.

Exercise Profile

- It should be the same for all exercises.
- It could be autogenerated based on the JSON Schema and fully extensible.

Exercise Slides

- These need to have a high level of customizability (having various types of slides).
- Individual tools can leverage the schema for their work in a limited way.

■ 3.7.2 Extensibility of the Code

The most important aspect influencing the maintainability and extensibility of any code is how it is structured and formatted and its readability level. As Robert C. Martin says: „The functionality that you create today has a good chance of changing in the next release, but the readability of your code will have a profound effect on all the changes that will ever be made. The coding style and readability set precedents that continue to affect maintainability

and extensibility long after the original code has been changed.” (Robert Cecil Martin, 2009)

In particular, the application code of the Exercise Designer should:

- Make it easy to add new tools. Business logic and components specific to each tool should be clearly decoupled from the other tools and from the rest of the system. New tools should be able to follow the existing patterns and plug in the current code.
- Offer a clear hierarchical project structure, making it easy to locate key parts of the system.



Chapter 4

Design

Design is „a specification of an object, manifested by an agent, intended to accomplish goals, in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to constraints” (Ralph and Wand, 2009). This specification describes the basic architectural parts of Exercise Designer, how they fulfil the requirements, and presents an initial design prototype.



4.1 Application Architecture

Exercise Designer application will follow a client-server model, which is very common in web applications. It will be composed of three parts: front end, back end, and the database. React application on the front end should satisfy most of the requirements. However, a back-end server and database are still needed to offer users a level of persistence - the ability to save and load their exercises during the editing process. Furthermore, the back end will allow users to generate a preview of their exercise, delegating to the HistoryLab application to generate a preview of the exercise.

■ 4.1.1 Front End

React application on the front end will be the central part of the Exercise Designer.

It will allow users to:

- Create a new exercise or load an existing exercise JSON file, modify it, and save the result while ensuring that the result exercises are compatible with the existing HistoryLab exercises.
- Navigate between different exercise slides and use various tools in order to build slides of different types. Users will be able to select the type of slide, and the tools will be available accordingly.
- Validate the exercise against the JSON schema. This is to ensure that all the required data have been filled and the exercise is ready for release.
- Upload new media files, which can be added to slides of the appropriate type.
- Store their intermediate progress to a database.
- Preview their exercise as if published in the HistoryLab application.

■ 4.1.2 Back End

A back-end server will be necessary for features that the front end cannot offer directly.

It will:

- Provide REST API endpoints for the front end.
- Communicate with the database in order to load or save the exercise.
- Leverage HistoryLab application's system of exercise generation and generate a preview of an exercise, then provide it to the front end.

■ 4.1.3 Database

A very simple NoSQL database will be used to store the exercise JSON files. A document store with support for JSON documents should be the best fit. Figure 4.1 shows a proposed schema of a single collection to be used for this purpose. `isDraft` property can be used to track the state of development of the exercise.

| Exercise | | |
|----------|-----------|----------|
| PK | id: | int |
| | exercise: | Document |
| | isDraft: | boolean |

Figure 4.1: Proposed Database Schema

■ 4.2 Exercise Tools

Each slide in an exercise is composed of one or more exercise tools. These tools offer various predefined ways of interaction with the exercise and can be considered basic building blocks of the exercise. Finding the best possible representation for these tools is one of the requirements.

■ Exercise Structure

- Each exercise is composed of one or more slides of various types (see illustration in Figure 4.2).
- Each type of slide defines a set of interactive tools and how these are used within the slide.

HistoryLab application defines which tools can be used together in a single slide, how they are visually rendered, and how their interaction is handled. As such, Exercise Designer has to follow these predefined patterns and should work with the basic abstraction unit of 'slide' of a particular type instead of allowing a user to work with each individual tool directly. These individual tools should still be defined independently on the level of application components or modules. But these components will be grouped in a tree-like fashion as part of 'slides' of various types.

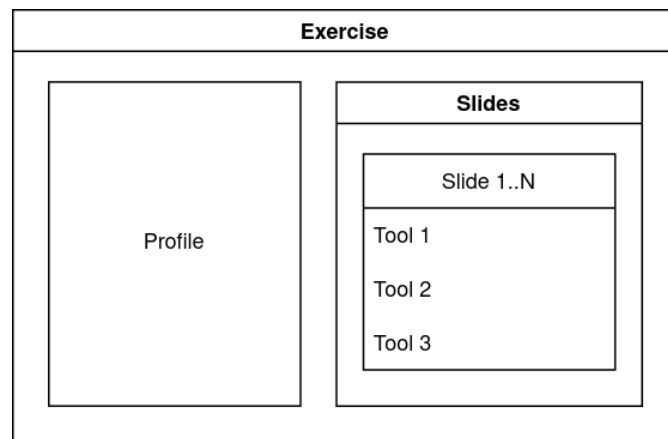


Figure 4.2: Exercise Structure Illustration

Exercise Designer should be designed in a modular fashion to guarantee extensibility:

- New slide types should be easy to create and should allow for easy integration of the existing tools.
- New tools should be easy to create and should allow for easy integration with the existing slide types.
- Implementation of individual slide types should be considered a black-box abstraction. Exercise Designer „should not need to know "how" each tool works, only that it does” (Abelson, 1996).
- This modular system should have conceptual integrity (Brooks, 1975) and coherent design.

■ 4.3 Design Prototype

Main Content. Represents the part visible to users in the HistoryLab application.

- Design similar to the actual HistoryLab application.
- Elements editable using input fields as well as mouse interactions (drag and drop).
- Navigating between slides using Previous and Next arrows in the top part.

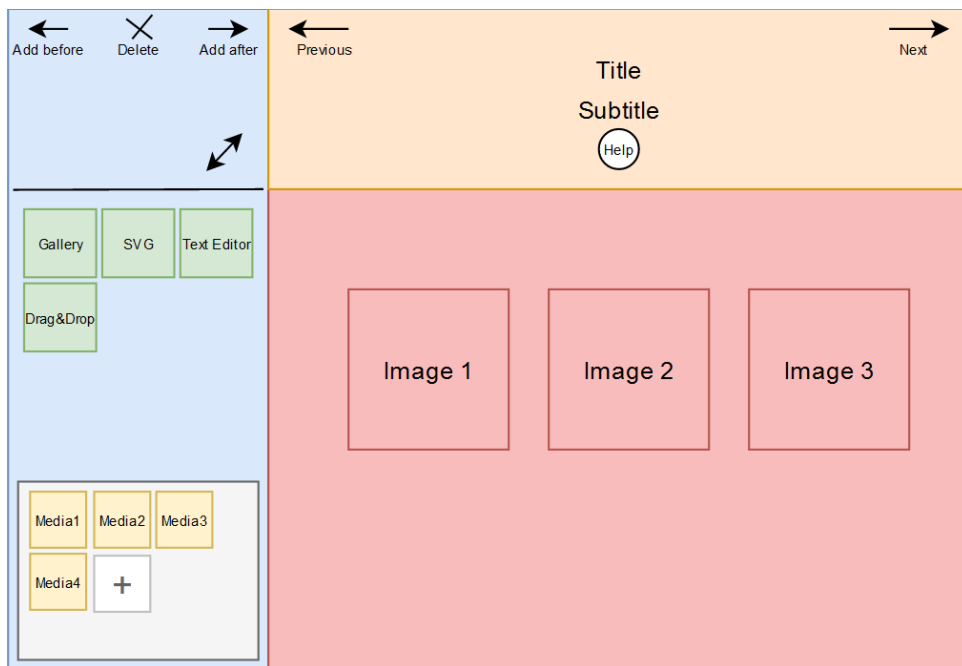


Figure 4.3: Exercise Designer Visual Prototype

Blue Sidebar. Contains all editor tools.

- The top part is a toolbar with add/remove slide buttons and other slide configuration buttons.
- The middle part is a library of slide types, which can be dragged and dropped over to the main content to select the type of slide.
- The bottom part is a media library. It is possible to add media and drag and drop the media items to the main content.



Chapter 5

Implementation



5.1 Overview



5.1.1 Front End

Front End application is the core of Exercise Designer, with most of the functionality implemented here. The application is split into two distinct parts - the profile editor and the slide editor. This separation mirrors the basic structure of each exercise - exercise metadata and a list of individual slides.



Profile Editor

Profile editor (see Figure 5.1) uses the original idea of form generation based on JSON Schema. This part of the exercise, where the user can fill in the metadata of the exercise, is the same for all exercises, and there is not much need for customization. To achieve the best extensibility using purely the JSON Schema, profile editor is composed of a dynamically generated form.

Figure 5.1: Profile Editor

Slide Editor

Slide Editor (see Figure 5.2) allows users to modify the contents of individual slides of the exercise. This part of the Exercise Designer uses a combination of statically and dynamically defined sections and components. Users can edit the exercise contents by adding, removing or editing the individual exercise slides. Each slide type defines which tools are available in the slide, and the media library allows users to upload media files.

Figure 5.2: Slide Editor

■ 5.1.2 Back End

Backend is a simple NodeJs Express application. It is responsible for communication with the database and for generating exercise previews. It also provides a simple web server that hosts the generated exercise previews.

■ 5.1.3 Database

Mongo DB document store with a single collection 'Exercise' has been used to store the exercise data.

Collection Schema:

```
{
  "title": "Exercise",
  "properties": {
    _id: { type: String, required: true },
    name: { type: String, required: true },
    value: { type: String, required: true },
    isDraft: { type: Boolean, required: true }
  }
}
```

■ 5.1.4 Interactions

Figure 5.3 shows some basic sample interactions between the user and the individual parts of the system. In particular, it shows that the back-end server is used both by the front-end application and by the actor directly to access the generated previews.

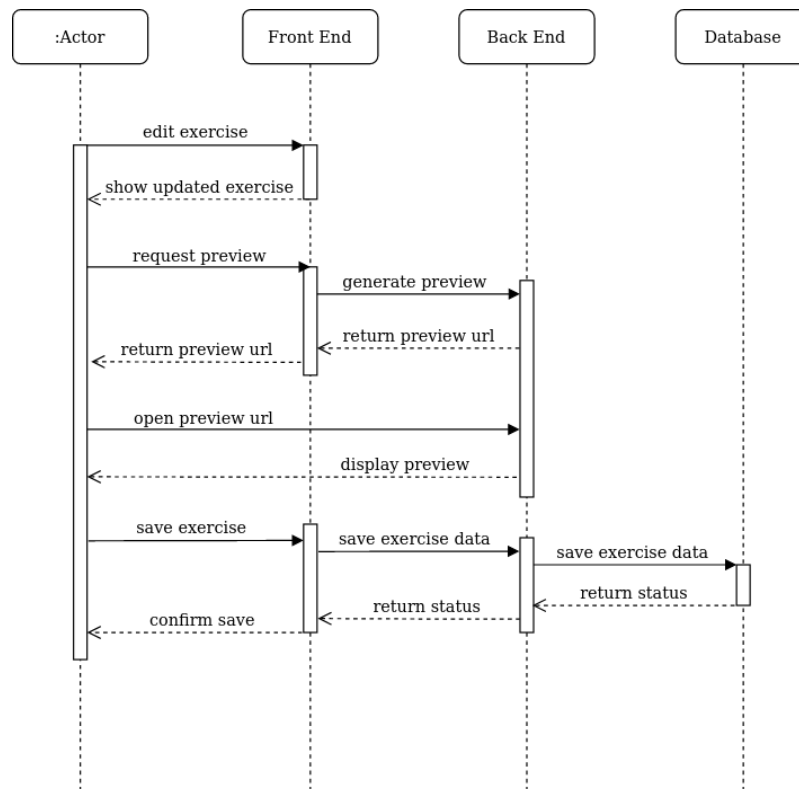


Figure 5.3: Sample Interactions

5.2 Key Technology

Typescript

TypeScript is a strongly typed programming language that extends Javascript. It adds many features of object-oriented languages, making it easier to structure code, write cleaner and self-explanatory code, and it provides a level of code quality assurance with its static type checking. (Microsoft, 2015)

Redux

Exercise Designer uses the Redux library for its state management. Redux enables the application to have a single state accessible throughout the application. This streamlines the workflows where different application parts need to interact with different parts of the editor state.

Redux state is accessible to the components using Selectors from the Reselect library. Selectors are simply memoized functions that provide access to a particular part of the Redux state, possibly combining different parts of the state. Being memoized, these functions return pre-calculated results and only recalculate when any of the inputs (the application state) change.

One of the challenges of implementing centralized state management using Redux was to guarantee the extensibility for new tools and the decoupling of existing tools. This was solved by forcing all tools to define their own Redux actions, reducers and selectors, which get registered with the global state management.

For example, React component generating the 'keywords'-type slide dispatches Redux actions for all user interactions that change this slide's contents, such as adding or removing a keyword or editing a keyword text. Each action is then handled by a Redux reducer, which decides how the slide's state changes as a result of this action. Adding a keyword may simply add a new item to the keywords data array. After the Redux state has been updated, React component will get notified by its subscribed selectors and will update its internal state and view. Keywords selector will return an updated list of keywords, the view will re-render, and the new keyword will appear on the screen.

■ Redux Saga

Redux Saga¹ is a side effect manager for Redux. It implements the Saga pattern to handle complex or asynchronous actions. Its advantage is using Javascript's generator functions, which allow for high readability and testability.

¹<https://redux-saga.js.org/>

■ 5.3 Exercise Tools

■ Each tool

- Is represented by a React component or a tree of components.
- Defines its own set of interactions with the exercise data structure.

■ Extensibility

1. Add the new slide type or tool to the JSON schema.
2. Create its respective React component and Redux integrations (reducers, actions, selectors).

■ 5.4 Implementation Details

■ 5.4.1 Schema Conversion

Exercise Designer works with a modified version of the exercise JSON Schema. To ensure interoperability with the HistoryLab application, conversion between the original and this new version of exercise JSON is necessary. Exercise Designer integrates a custom Schema Data Converter, which handles bi-directional conversion of exercises between these two versions of JSON schema.

The conversion is implemented using a Production Rule System. „The Production Rule System computational model implements the notion of a set of rules, where each rule has a condition and a consequential action. The system runs the rules on the data it has through a series of cycles, each cycle identifying the rules whose conditions match, then executes the rules' actions.” (Fowler, 2021).

■ Transformation Rule

Transformation Rule is simply „a rule in a Production Rule System. Such a rule has two parts: a Boolean condition and an action” (Fowler and Parsons, 2011). This action specifies how the data is transformed when the condition is true. Its subclasses define specific rules for transformation (eg. `RenameObjectKeyRule`, `DropObjectKeyRule`). Rules can be either stateless or stateful, where stateful rules leverage an additional action that gets triggered when leaving the node (after the entire subgraph has been traversed). Rules can be further parametrised to reduce the number of subclasses (it is possible to define a very general rule that expects a parameter or a very specific rule that does not need one).

■ Ruleset

Ruleset represent a collection of rules with the purpose of transitioning from one state to another. Its subclasses specify concrete collections of transformation rules that are used for the conversion (e.g. `OldToNewRuleset`, `NewToOldRuleset`). This collection implements the Iterator design pattern (Freeman and Robson, 2014) to allow iterating over the individual rules.

■ SchemaDataConverter

`SchemaDataConverter` is responsible for the conversion of the data. It receives a concrete transformation ruleset along with the input JSON data. It iterates over the Ruleset, and using depth-first search, it recursively traverses over the input data structure and tries to apply the transformation rule. The result is the transformed JSON data.

■ 5.4.2 Schema Representation

JSON Schema representation is simply a JSON object, and as such, it may be difficult to get the required information from such a data structure. It is important to be able to work with the schema in an effective way not only to generate the dynamic form but also to be able to access other key schema



Chapter 6

Testing

In the development cycle of any application, testing is one of the key aspects that need to be present. Testing makes sure that the application is being developed with a level of quality in mind and gives some confidence to developers who know that their code works as expected.



6.1 Static Analysis

„Static analysis is analysis of the structure and text of a program without executing it” (Boulanger and Boulanger, 2011). It is the first line of defence that notifies the programmer if there is any problem with the code. In particular, static analysis helps find issues with the syntax, code that does not follow the preset coding standards of the project, and can even discover some fundamental semantic problems with the code.

Exercise Designer uses two static analysis tools:

1. TypeScript. TypeScript itself provides a simple static analysis.
2. ESLint. ESLint is an open-source linting tool. It is highly customizable, supports different languages and different coding rules configurations.

■ 6.2 Unit Tests

Unit tests are tests on the lowest abstraction level of development. They test 'units' of code, which are usually individual functions or groups of functions doing a single thing.

Exercise Designer has only a few sets of unit tests. In particular, these tests test the schema conversion and validation.

■ 6.3 API Testing

API tests are tests executed against an API to ensure that it accepts requests as expected and returns the expected results.

Exercise Designer back-end server provides a simple REST API. This API is tested using the Postman platform.

■ 6.4 User Testing

■ 6.4.1 Methodology

„For user testing, the finding of a usability problem depends on two factors: First, the subject has to experience the problem, and second, the experimenter has to realize that the user experienced the problem.” (Nielsen and Landauer, 1993)

Firstly, in order for testers to experience the potential problems, I have created a list of tasks to be performed in order on the system under test. These tasks would have the testers proceed through all key parts of the system.

Secondly, to be able to get feedback, testers were instructed to comment

on how easy or difficult it was to perform each of these tasks, as well as any other thoughts or opinions they had while performing these tasks.

As for the number of testers, a research study by Nielsen and Landauer (Nielsen and Landauer, 1993) found that the maximum cost/benefit ratio is reached with only four to five testers. Six people tested the application, including developers, didacticians and one QA specialist.

6.4.2 Tasks

Modify the pre-loaded exercise "What they expected of the sewing machine?" as follows:

1. Add years 1917 and 1924 to the timeline.
2. Insert slide of type **Sources & Descriptions** as a 3rd slide in order. Fill in the main and extended task description, add an image from the gallery along with a description.
3. Rename the exercise to „What the sewing machine brought to us?“
4. Add yourself as an author and change the exercise difficulty to hard.
5. On the second slide, replace an image with another image from the gallery, modify the description accordingly.
6. On the keywords slide type, remove keywords **poverty** and **work**, add a keyword **education**.
7. In the text editor, add the option to highlight in blue parts of the text which describe the purpose of a sewing machine. Modify the description accordingly.

Create a new exercise as follows:

8. Fill in all the mandatory parts of the exercise profile.
9. Add three slides of the following types in the same order: **Sources & Descriptions**, **Keywords**, **User Text**.
10. In each of these slides, fill in the main and extended task description and add at least a single element to the slide content.
11. Upload your own image to the media gallery. Add it to the content of slide type **Sources & Descriptions**.
12. Split slide of type **Keywords** so that the left part of the content will show a preview of the previous slide.

13. Add slide of type **Text Editor** to the end of the exercise. Add an image, text and keywords to highlight.
14. Delete slide of type **Keywords**.

Create a new exercise as follows:

15. Create your own exercise with at least 3 different types of slides. Fill the exercise with data.

■ 6.4.3 Results

Tester feedback to provided functionality was generally positive. Most testers were able to perform most if not all of their tasks. I have created a list of all issues and split them by their perceived priority (based mainly on the severity and the number of times the subject had appeared in the feedback).

High Priority.

- Schema validation. When schema is invalid, the reason is not communicated to the user.
- Number field behaviour. Zero digit (0) in number fields cannot be deleted.
- Buttons need visual improvements. Buttons to add slides are easy to confuse with the navigational buttons. The button to split the current slide is hard to find and needs a label.
- Slide type information. It is hard to tell the type of current slide or slides in the split layout.

Medium Priority.

- Unpolished field labels. Some field labels have array-like numeric indexes, and some contain object-like dots.
- No focus on the added field. This field should automatically focus when adding a new field in the profile.
- Color picker is not needed.
- Some profile fields should offer a selection from a list. **Keywords** and **epoch** fields should be restricted to a selection of pre-approved words to pick from.

- Some profile fields should not be editable. `global` and `content` version should not be editable.
- Text profile fields should adapt their length to their content. `Annotation` field does not show the full text on the screen.
- Some fields should be autogenerated. Profile `id` field should autogenerated with a unique value. Functions could be autogenerated based on the slide types.
- Minor technical issues. Help tooltip is showing `
` tag character. Help tooltip is incorrect for `text editor` slide type.
- How to reorder gallery images. This should be communicated better to the user.
- `Text editor` command field improvement. It should not support diacritics in its input. The value should be autogenerated if possible.

Low Priority.

- Responsiveness. Exercise Designer is too wide and not responsive.
- Back button function. It is not possible to revert the last change.
- Ability to change the type of slide.
- Split-layout visual improvements. Emphasize which slide is currently active.
- `User text` slide-type improvements. There should be a parameter called `instruction`. `Zadani...` should be a placeholder for the field, not the contents. The user should be able to set the minimum and ideal length of the text answer.
- `Keywords` slide-type improvements. It should be possible to reorder the tags. The plus button has a strange visual placement.
- `Drag & drop` slide-type improvements. There is a strange background colour issue.

Out of Scope.

- Add different media types to the gallery. (Currently, none of the implemented tools use media types other than image. These will be added in the future.)
- Limit the number of keywords that the student can select. (This is not implemented in the HistoryLab application.)

6.4.4 Changes in Response to Users' Feedback

Functional Improvements

- Number fields updated. Zero digit (0) value can now be deleted as expected.
- Type of the current slide is now displayed as a label at the top of the slide.
- In `text editor` slide type, help icon's tooltip text was updated.
- `Keywords` fields values are now selectable from values predefined in the schema.
- `Epoch` schema definition was updated, and field values are now selectable from values predefined in the schema.
- In the profile, `version` fields and `id` fields are now read-only. `Id` value is randomly generated for new exercises.
- In the profile, the newly added field is now focused automatically.
- Schema validation section now has a tooltip listing all the current validation errors.
- Color picker was removed from the profile.
- In the slide of type `user text`, the text field was renamed to `instruction` to avoid confusion. `Zadani . . .` text was moved into a placeholder. Three fields were added in order to set the minimum, optimal and maximum length of the answer.
- In the slide of type `text editor`, the command field is now autogenerated from the name field and does not support diacritics input.

Visual Improvements

- Improved responsiveness. Labels in the top bar will now resize on smaller screens. Exercise title will now not overflow its container. However, at least 1000px screen width (desktop) is still highly recommended.
- Cleaned up field labels. Numeric indexes have been removed, and diacritics have been added as needed.
- All text fields in the profile will now adapt their length to the content and print the content in multiple lines if needed.

- Improved buttons. **Split** button now has a text label and a button-like shape. Add-slide buttons now have a plus-sign icon, rather than an arrow icon, to avoid confusion with the navigational buttons. Navigational buttons' design was updated to be more button-like and easier to notice for the users.
- Split slide's preview part is now partially transparent with a watermark to emphasize that this is the previous slide.
- Placement of the **keywords**-slide plus button has been updated to fit better the layout.
- Gallery image styles updated to fix background colour transition issue and make reordering images easier to understand.



Chapter 7

Conclusion

■ Limitations

Exercise Designer needs to be backwards compatible with the existing exercises. Currently, there are over 150 existing exercises; each is composed of several slides, and each slide is composed of a number of tools. How these tools are organised and interact with each other depends on the type of each slide. Implementation of these tools in the HistoryLab application is complex, featuring various options and nuances. As such, this project's scope could not cover all of the functionality and had to be reduced. Only 6 out of the existing 12 tools have been implemented, some with only partial functionality. However, the current Exercise Designer can be considered a base platform, with the potential to be easily extended in the future to include all of the tools.

■ Summary

Exercise Designer application has been created based on analysis of the requirements and research of the existing situation. HistoryLab exercises were analysed in order to create the best possible representation for the individual tools. Based on this analysis, appropriate software architecture and tools were chosen in order to implement the application. Exercise Designer has been created as an interactive web application.

Users of Exercise Designer can use a variety of tools to build a new exercise or modify an existing one. They can create draft exercises and quickly test if such exercises match their expectations. This should help reduce the communication overhead described in the motivation section.

■ Future Improvements

Due to the limited time and resources available for this project, the scope of the work had to be restricted to fundamentals in order to meet all the relevant due dates.

Support More Tools. Exercise Designer should be extended to support all the tools available in the HistoryLab application. This includes the ability to upload different media types to the media library.

'Go Back' Feature. Users should be able to revert their most recent changes.

Cache Generated Previews. The application should 'remember' which previews were already generated, and these should be available quickly.

Add More Tests. More automated tests should be added to „guarantee developer productivity and quality of code” (Osherove, 2006).

Other Minor Improvements.

- Make it possible to rearrange the tags in **keywords** type of slide.
- Add the ability to change the type of slide.



Bibliography

- Abelson, H. (1996). *Structure and interpretation of computer programs, second edition*. The MIT Press.
- About Trello. (n.d.). Retrieved from <https://trello.com/about>
- Boulanger, J., & Boulanger, J. (2011). *Static analysis of software : The abstract interpretation*. John Wiley & Sons, Incorporated. Retrieved from <http://ebookcentral.proquest.com/lib/cityuseattle/detail.action?docID=1124674>
- Brooks, F. (1975). The mythical man-month (1975). doi:10.7551/mitpress/12274.003.0042
- CWN. (2016). *Computer Weekly News*, (2183), 2183. Retrieved from https://link.gale.com/apps/doc/A450431907/GIC?u=cityu_main&sid=summon&xid=e0472c8b
- Fowler, M. (2021). Production rule system. Retrieved from <https://martinfowler.com/dslCatalog/productionRule.html>
- Fowler, M., & Parsons, R. (2011). *Domain-specific languages*. Addison-Wesley.
- Freeman, E., & Robson, E. (2014). *Head first design patterns : A brain-friendly guide*. O'reilly, Edition: 10Th Anniversary Ed.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Pearson Education. Retrieved from <https://books.google.cz/books?id=6oHuKQe3TjQC>
- Greif, S., & Benitte, R. (2021). State of js 2020. Retrieved from <https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>
- Hall, H. (2019). Enhance enum support (defaults and unsetting) · issue #1501 · eclipsesource/jsonforms. Retrieved from <https://github.com/eclipsesource/jsonforms/issues/1501>
- HistoryLab. (2021). Retrieved from <https://historylab.cz/en/>



Appendix A

DVD

Attached DVD contains the following:

- This document in the PDF format.
- Spreadsheet with the original test results.
- The application source code.

■ Application source Code

`/source-code` folder contains the source code for the Exercise Designer.

- `/scripts` — scripts for conversion of exercise JSON to a different schema
- `/server` — back end source code
- `/src` — front end source code
- `README.md` file describes how to run the project.
- `package.json` file contains the NPM project manifest.



Appendix B

User Testing Results

The test assignment was given to 9 independent testers (labeled A to I). However, testers D,E and H did not participate in the testing and their results were not included.

Feedback from the testers was split into two categories:

- Visual issues
- Functional issues

■ Tester A

Functional Issues:

- In the profile editor, I do not like that the 0 value is the default for number fields and that it cannot be erased.
- Global version should not be editable. It is based on the version of the application.
- Content version should probably not be editable either — every publication of exercise should automatically increase the value by 1.
- `Keywords` and `epoch obdobi` values should be selected from a predefined list.
- I would expect autofocus on the newly added input field.

- Functions could be added automatically, based on the exercise slides.
- Exercise ID should generate automatically, and it must be unique.
- In the `text editor` slide type, I do not know what the purpose of the `command` field is.
- In the `text editor` slide type, it should be possible to set the minimum and ideal length of the answer.
- In the `svg` slide type, it should be possible to select the function - drawing, creating dots or adding comments
- In the `gallery` slide type, it should be possible to add previous work results (SVGs), texts, audio or video files.
- The application says that the schema is not valid, but it doesn't say why. It would be nice to have a list of errors.

Visual Issues:

- Annotations are long texts, and the text fields should be bigger to accommodate this.
- I would remove all array-like numeric indexes in labels.
- To add a new slide, I would replace the current arrow icons with a plus symbol. Arrows are associated with moving, and I would expect them to be used for navigation.
- The `split` button needs a visible label, and the toolbar design should be improved.
- In the `keywords` slide type, plus button to add a tag has a strange visual placement.

■ Tester B

Functional Issues:

- I can not remove value 0 in the year and duration fields in the profile editor.
- It is not good to remove images from the media gallery after being used. Teachers may want to use the same image on the same slide again.
- In the profile editor, it would be nice to have it automatically be in focus after adding a field.

- In the profile editor, oftentimes the placeholder is [0] or [1]. It would be better to remove this, or replace with something better.
- Help field tooltip shows `
` tag characters.
- **drag and drop** slide type has a strange background color transition (accompanied by an image).

■ Tester F

Functional Issues:

- It is not clear what is the purpose of colour selection.
- It would be nice to see the type of current slide somewhere. The help icon is not enough.
- It would be nice to be able to change the type of current slide.
- It would be nice to have a 'go back' function to prevent accidental changes (including slide deletions).
- I cannot change the order of tags in **keywords** slide type.
- In the profile editor, keywords and epochs should be limited to a selection of pre-approved words.
- In the **keywords** slide type, is it possible to limit the number of keywords, which the student can select?

Visual Issues:

- For the split-screen slides, it may be less confusing just to show the icon, not the previous slide itself.
- In the profile editor, it was hard to understand what is what. Can the visuals be improved?
- In the profile editor, the annotation labels should be "Ucitelske" and "Zakovske", not "Verejne".
- It is unclear how to add a caption to a gallery image.
- It is not possible to change the order of images.

Appendix C

Images

These are high-resolution slides of the exercise "What they expected of the sewing machine?" and the Exercise Designer's user interface.

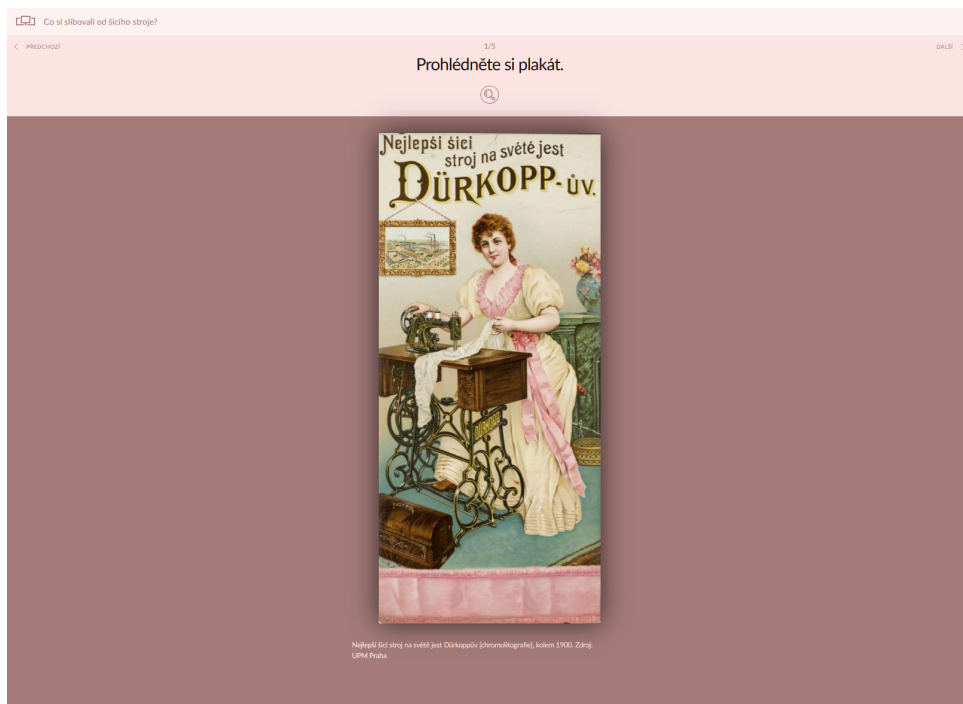



Figure C.1: Example Exercise - Slide One

C. Images

Co si slibovali od šicího stroje? 2/5

Jak plakát představuje šicí stroj?
Označte důležité prvky na plakátu a popište je.




Nejlepší šicí stroj na světě jest Dürkoppův (chromolitografie), kolem 1900. Zdroj: UPVH Praha

Figure C.2: Example Exercise - Slide Two

Co si slibovali od šicího stroje? 2/5

Jak plakát představuje šicí stroj?
Označte důležité prvky na plakátu a popište je.



Nejlepší šicí stroj na světě jest Dürkoppův (chromolitografie), kolem 1900. Zdroj: UPVH Praha

3/5

Jak plakát představuje šicí stroj?
Vyberte na základě plakátu maximálně tři slova, která nejlépe vystihují význam šicího stroje.

bohatství chudoba pohodlí volný čas práce
domov stroj pro ženu

Figure C.3: Example Exercise - Slide Three

Co si slibovali od šicího stroje?

4/5

← PŘEDCHOZÍ

Jak se psalo o šicím stroji?

Přečtěte si text a označte žlutě pasáže, které jsou věnovány vlastnostem šicího stroje. Červeně vyznačte pasáže, které označují, komu je stroj určen.

1



2

Záhodno najisté jest zmíniti se o stroji blahodárném, jenž za naší doby vynalezen býti, praktičností a užitečností vyniká i úpravou odolností se odznáčuje a za každou ruku sloužit může, okrajového hospodářství komnatu. Jest to šicí stroj, pomocí kteréhož možno jest hospodyně samé, zaměstnané rozsáhlým hospodářstvím, zhotoviti v přetráčkém čase všecko, cokoliv v domácnosti šiti potřeba jest, tak, že i při větší rodině švadlena zbytečná se stane. Ač obdržeti lze rozličné šicí stroje i z rozličných továren, jsou přece americký strojové toho druhu nejlepší, jsouce zřízeny ne toliko k oborjímému sešívání, nýbrž i k obrábování, započívání, věšání a prořívání šitinek a jejich odolností. Takový stroj byl by zajisté vítaný ve vědě nevědy, jelikož náklad za něj učiněný za krátký čas se vyplácí.

3

František Hanuš, Frisco. Hospodářské radobce věku: krátké naučení o vedení měšičkého i verkovského hospodářství, Praha 1874

Figure C.4: Example Exercise - Slide Four

Co si slibovali od šicího stroje?

5/5

← PŘEDCHOZÍ

Odpovězte na otázky.

1



Co si slibovali od šicího stroje?

Myslím si, že...

Kdo a k čemu podle vás používá šicí stroj dnes?

Šicí stroj používá...

Figure C.5: Example Exercise - Slide Five

☰ Editor cvičení 1 Editor profilu 2 Editor slajdů

Co si síbavali od šichho stroje?

DB Status: ✓

validní schéma

Draft

GENEROVAT NÁHLED

Editor profilu

Version

Global

<%= version %>

Katalog

test

Language

cs

Slug

co-si-sibavali-od-sichho-stroje

Content

1

Autor

Václav Sixta

Nazev

Co si síbavali od šichho stroje?

KlicovaSlova

Rvp

technika

Koncept

gender

Historylab

formulujeme a ověřujeme hypotézu

CasovaOsa

Roky

1900

Obdobi

habsburská monarchie a její rozpad

Epochy

Refid

czech

Obdobi

Modernizace Evropy a rozmach koloniálních říší

Refid

world

Obdobi

Modernizace Evropy a rozmach koloniálních říší

Figure C.6: Exercise Designer - Profile Editor (Example Slide)

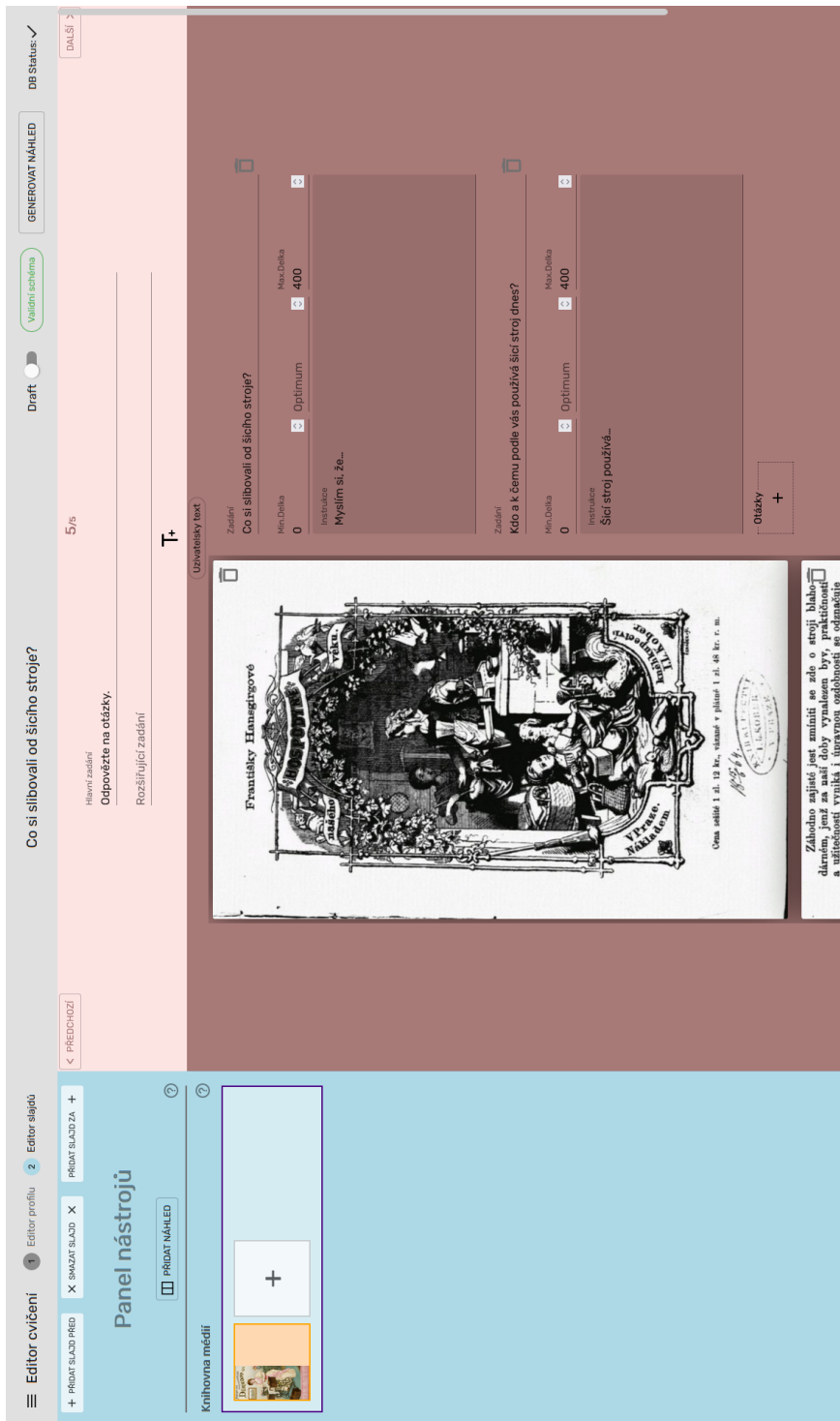


Figure C.7: Slide Editor