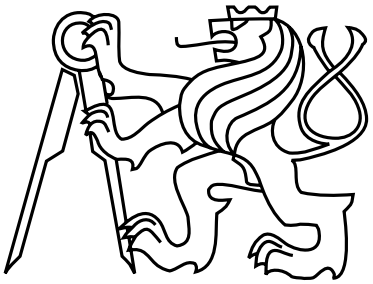
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor's Thesis

Web Application for Text to Voice Form Transformation

Maksym Polovnikov

Supervisor: Ing. Václav Chudáček, Ph.D.

Study Program: Open Informatics

Branch of Study: Computer and Information Science

May 22, 2019

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Polovnikov Maksym** Personal ID number: **452769**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Web Application for Text to Voice Form Transformation

Bachelor's thesis title in Czech:

Webová aplikace pro převod textového formuláře do hlasové podoby

Guidelines:

1. Review existing solutions for form representation.
2. Review existing chat solutions (voice, text).
3. Design a web application for design of forms with validation, UI and user management.
4. Implement the application and provide an export for automated creating of dialogs (YAML).
5. Test on existing (DGI) chat platform.
6. Test and document the application.
7. Optionally, implement demonstration example.

Bibliography / sources:

[1] HTML Standard https://www.w3.org/TR/html/
[2] Jurafsky and Martin, Speech and Language Processing (online avail. http://web.stanford.edu/~jurafsky/)
[3] ECMA-262 stanadard, https://www.ecma-international.org/publications/standards/Ecma-262.htm

Name and workplace of bachelor's thesis supervisor:

Ing. Václav Chudáček, Ph.D., CIIRC ČVUT v Praze

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **12.01.2018** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **22.09.2019**

Ing. Václav Chudáček, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgement

I would like to thank Ing. Miroslav Burša, Ph.D. and Ing. Václav Chudáček, Ph.D. for their expert guidance, provided literature and language correction.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 22.05.2019 .....

Abstract

The aim of this work is to design, implement and test a web application for creating forms and filling them using voice devices, such as Amazon Alexa or using chat platforms such as Slack.

The web application was implemented using ReactJs as front-end platform and Redux server-side library. It has been tested on Alexa for Android and Alexa Echo dot device.

This application allows a user to create an HTML-like form with element validation, for example : business trip insurance, food ordering, cinema ticket reservation, etc.

In this thesis we have described the process of designing, implementing and testing the application. We have also included a detailed workflow of dialog creation together with an example of user interaction.

Abstrakt

Cílem mé bakalářské práce bylo navrhnout, implementovat a otestovat webovou aplikaci pro vytváření a správu webových formulářů, které mohou být přetransformovány do podoby hlasové nebo dialogové aplikace (např. pro platformy Slack nebo Amazon Alexa).

Front-end část webové aplikace je implementována pomocí technologie ReactJs, serverová čast používá knihovnu Redux. Vytvořené dialogy byly úspěšně otestovány na zařízení Alexa Echo Dot a pomocí programu Alexa for Android.

Aplikace slouží k vytváření formulářů (podobných HTML formulářům), například pro potřeby pojištění služební cesty, rezervace lístků do kina, apod. Jednotlivá pole formuláře je možné ověřit (validovat).

V této bakalářské práci popisuji postup návrhu, implementace a testování popisované aplikace. Je také uveden podrobný postup vytváření vzorového dialogu spolu s textovým přepisem hlasové interakce s uživatelem.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Jurafsky in his book "Speech and Language Processing" [8] identifies two main algorithms classes of conversational agents.

Task-oriented dialog agents are designed for a particular task and set up to have short conversations to get information from the user to help complete the task. Companies deploy goal-based conversational agents on their websites to help customers answer questions or address problems. Conversational agents play an important role as an interface to robots.

Chatbots are systems designed for extended conversations, set up to mimic the unstructured conversational or 'chats' characteristic of human-human interaction, rather than focused on a particular task like booking plane flights.

In today's world, every day the impact of conversational agents is becoming more and more. The rhythm of human life of the 21st century has accelerated and now things that have not previously influenced the human schedule, become time-consuming. This is what happens now with Web technologies, most programs or Web services try to offer their users a more convenient way of communication, namely the Voice assistant. Over the past 5 years, thanks to companies like Google and Amazon, the scope of using voice assistants has expanded from an ordinary task to switch to the next song to maintaining a dialogue with the user on any topic and performing more difficult tasks. This work presents a way, how a user, with the help of Task-oriented dialog agents Alexa, can fill the custom created form.

1.1 The purpose of the bachelor thesis

The task of the work is to implement a web application for designing and creating HTML-like forms and transformed into voice form. When the form is created, the application will have the functionality to create a form dialog and upload the dialog representation to dialog system (DGI, Dialog Graph Interpreter). The outcome can be used by Alexa [2] (or other voice) platform. The form would have standard fields (number, date) and field for help information.

It can be used for example for:

- Plane ticket reservation

- Pizza order
- simple shopping list or other

1.2 Objectives

The main objectives of this work are :

1. Review existing solutions of form representation
2. Review existing chat solutions (voice, text)
3. Design a web application for design of forms with validation [1], UI and User management
4. Implement the application
5. Provide an export for automated deployment of dialogs (YAML)
6. Test on existing (DGI) chat platform
7. Test and document the application
8. Implement a demonstration example

Chapter 2

Requirements

2.1 Functional requirements

The functional requirements specification describes the expected functionality.

1. Create Form
   (a) Add/Remove input item (Modify)
   (b) Set item id (Mandatory, unique)
   (c) Set help (Optional, gives additional information about item) [1]
   (d) Add item sub-fields
2. Clear generated form
3. Process filled form (data)
4. Export to YAML
5. Upload YAML to DGI (via REST API)
6. Set validation for item fields. (i.e required field, default values)
7. AWS Lambda function to provide dialog processing

Those aspects are not related to my work, but are a part of the whole system:

1. Test with Alexa
2. Deploy AWS Lambda
3. Deploy DGI

[1] An example :
user says: Help or I don't understand.
Possible help: Please, say the departure date, such as November twelve.

2.2 Non-functional requirements

Non-functional requirements are requirements that specify criteria that can be used to decide the correctness of the operation of a system, rather than specific behaviors.

1. Localization: only en_US
2. Browser portability
3. Reusability

Chapter 3

Technology overview

3.1 Voice response system

Voice response system (VRS) is a type of speech synthesis application which is used to create a spoken sound response. Voice apps become a common thing in our world and are used every day [3]. Unlike a text to speech system (TTS), where speech synthesis is used to create spontaneous sentence or phrase based on human phonetics, VRS operates with limited vocabulary in situations where the sentence or phrase that is formed follows a strict predetermined pattern.

3.2 Form builder

Form builder (FB) is a general name for applications, the main purpose of which is to provide a way of creating custom forms. There are existing lots of form builder applications such as Google Forms, JotForm, Alexa form builder and so on, but none of this can provide full functionality that is needed for completing the aim of this work

FB is an integral part of this work. Thanks to this, user could create a specific form, that will satisfy his requirements or modify already existing forms.

Also, it's needed to correctly collect input fields, which was created by User in a form.

3.3 YAML

YAML (Yet Another Markup Language) is a human-readable data serialization language [6]. In purpose, it is similar to XML or JSON. The structure of YAML is defined by indentation and data are stored as a map containing keys and values associated with those keys. This map is in no particular order, so you can reorder it at will.

In this work, YAML will represent the way of dialog processing i.e contain particular steps of user interaction model.

3.4 Alexa

Alexa is a voice service from Amazon. Alexa powers devices like Echo Show, Echo Spot, Echo Dot, etc. The most important feature of the Alexa is off course it's hands-free voice control, which lets users ask questions without touching the device. Alexa listen to the voice commands and respond with appropriate responses to fulfill. Alexa is a superhuman assistant and can understand multiple languages such as English, French, German, Japanese. Thus Alexa can interact with millions of users from different countries without any language barrier problems.

3.5 AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running. With AWS Lambda, you can run code for virtually any type of the application or backend service - all with zero administration. AWS Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. All user need to do is supply his code in one of the languages that AWS Lambda supports.

Chapter 4

Platform review

Before choosing a web framework, firstly it's needed to determine what script language will be used in this work.

4.1 JavaScript vs Typescript

For web development, script language is an internal part. TypeScript (TS) and JavaScript (JS) are two widely known languages in the developing world [10].

Choosing the correct one script language can highly speed up the development process.

4.1.1 JavaScript

JavaScript is a cross-platform, object-oriented scripting language used to make webpages interactive. This means that in the browser, JavaScript can change the way the webpage (DOM) looks.

4.1.2 TypeScript

TypeScript is both a language and a set of tools. TypeScript is a typed superset of JavaScript that is at the end compiled to JavaScript.

In other words, TypeScript is JavaScript plus some additional features.

4.1.3 Comparison

	JavaScript	TypeScript
Early error detection	Glaring error in JavaScript will never be singled out for correction until run time	Static Typing allows to see it before run time
Large App Capabilities	Wasn't created for large applications	Has large app capabilities
Script	JavaScript is based on EcmaScript 6	TypeScript is a superset of EcmaScript

Table 4.1: Comparison of JavaScript and TypeScript.

Depending on requirements and personal preferences, I decided to select JavaScript by these criteria:

- No additional setup is required for JavaScript
- ReactJs is more suitable for JavaScript
- Less chance of unexpected errors

4.2 Selecting a Web Framework

Web frameworks exist to make building a website faster and easier. Typically, frameworks provide tools to cover the common CRUD entities. Also, they afford libraries to interact with a database, for creating components (such as Pop-up window or ToolTip window) to display your HTML pages and more.

4.2.1 Comparison

For choosing the web framework we will use the following criteria.

4.2.1.1 Components

All of the web frameworks are component based. This means that component gets an input, after internal behavior (function execution) it returns a rendered UI template as output.

4.2.1.2 ES6 vs ES5 vs TypeScript

Web frameworks have preferred script languages. ECMAScript is still primarily a scripting language for web browsers, but last several years, TypeScript is becoming popular.

4.2.1.3 Templates. JSX, HTML

JSX is an optional preprocessor for HTML-like syntax that will be compiled in Javascript later.

JSX is a big advantage for development, because you have everything in one place, and code completion and compile-time checks work better.

4.2.1.4 Framework vs Library

Libraries are the older concept and these are just collection of utility classes/methods your code calls upon to get some functionality.

Frameworks, on the other hand, contain some functionality or flow and calls your code to extend or make the flow a specific one.

4.2.2 Web Framework

As mentioned before, web framework is a software framework that is designed to support (and speed up) the development of web applications including web services, web resources, and web APIs [12].

Nowadays the most popular and powerful frameworks are Angular, Vue.js, ReactJs.

4.2.2.1 Angular

Angular is a TypeScript-based Javascript framework. Developed and maintained by Google [4], it's described as a "Superheroic JavaScript MVC Framework".

Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular is used by Google, Wix, weather.com, healthcare.gov and Forbes.

Angular is good for building highly interactive web applications.

4.2.2.2 ReactJs

React is described as "a JavaScript library for building user interfaces" [11]. React is also called as "the V in the Model-View-Controller structure". At the heart of all React applications are components.

A component is a self-contained module that renders some output. We can write interface elements like a button or an input field as a React component. Components are composable.

A component might include one or more other components in its output. React is used by Facebook, Instagram.

4.2.2.3 Vue.js

Vue.js is one of those new software technologies that are being widely used across the world for web development [5]. Vue.js is actually a JavaScript framework with various optional tools for building user interfaces.

4.2.2.4 Table of comparison

Criteria	ReactJs	Angular	Vue.js
Is component based?	Yes	Yes	Yes
ES6 vs ES5 vs TypeScript	ES6	TypeScript	ES5
JSX, HTML	JSX	HTML	HTML
Framework vs Library	Library	Framework	Library

Table 4.2: Comparison of ReactJs, Angular ana Vue.js

4.2.2.5 Selection

Every of frameworks has it's own advantages and disadvantages. Angular is the most powerful among all of them. Vue.js has the biggest level of flexibility. ReactJs in turn concentrates more on dynamism. Depending on my personal preferences in this work ReactJs will be chosen for the following reasons:

- ReactJs has compatibility with JSX
- This work is not supposed to be big, so ReactJs will be more suitable for developing such application.
- ReactJs has a high quality base of additional libraries

4.3 Redux

In order to avoid uncontrolled state jumps between components, we need a way to properly transfer states. It is hard to keep the track from where (which component) the data is coming from. In React there is one way data flow. React would pass down the data from first level component to second level component and so on, which is not the best practice and Redux solves this state transfer problem.

Redux is a concept of data storing and communication within an application in the client-side (in the browser) [7]. It is suitable for single-page applications where state management can become complex over time. Redux is not associated with any particular framework, but it was developed especially for React.

The data in the React is "flowing" through the components. This is called a "unidirectional data flow" – the data flow goes in one direction, from the parent to the child. It is not obvious how two components that are not related to the parent-child relationship will interact with each other.In React, it is not recommend to implement a component's direct interaction.

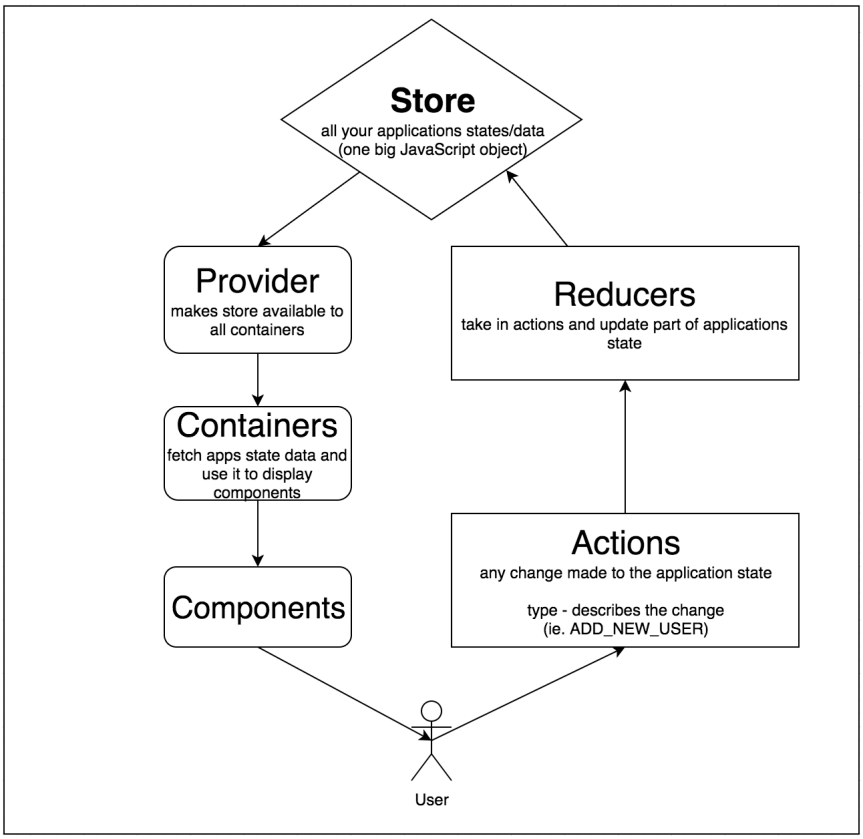


Figure 4.1: Redux data flow

Here handy Redux. Redux offers to store all application state in one location called "store". The components "Send" the state change to the store, not directly to other components. The components that should be aware of these changes are "subscribed" to the vault. (see Figure 4.1)

Redux can also be described like a backend's database on the client-side where developer store all the required information that is required in order to generate the view. It helps developer to to query SELECT, INSERT and UPDATE the data into the database.

4.4 JSON Web Token (JWT)

JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object [13].

The most common scenario for using JWT is an authentication. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.

JWTs can be signed using a secret or a public/private key pair. A big advantage of JWT is signed tokens. Signed tokens can verify the integrity of the claims contained within it.

JWT offers the following features:

- Compact. They are small, that's why JWTs can be sent through a URL, POST parameter, or inside an HTTP header.
- Self-contained. The payload contains all the required information about the user.

JSON Web Tokens consist of three parts separated by dots, which are:

- Header. It consists of two parts: the type of the token and hashing algorithm.
- Payload. It contains claims. Claims are statements about an entity and additional metadata.
- Signature. This part is a sign combination of the encoded header, the encoded payload, a secret, the algorithm specified in the header.

4.5 Jest

To perform unit testing of an application, we need a tool for this. Our application is based on React, because of that, we will use **Jest**, React-specific test runner.

Jest is the unit testing framework for ReactJS project. It is provided and used by Facebook themselves [9].

Other reasons for using Jest are:

- Automatically finds tests
- Automatically mocks dependencies
- Runs your tests with a fake DOM implementation
- Runs tests in parallel processes

Chapter 5

Application architecture overview

For the better understanding of how this application works it is needed to provide the full image of the application structure.

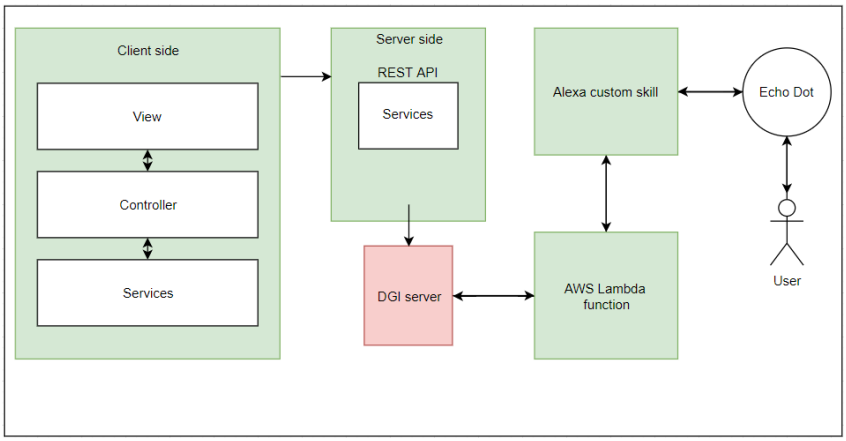


Figure 5.1: Application architecture

In the image above is introduced the full flow of the application. In the first step user will create custom form in the web application (Chapter 6). After submitting, client-side part (Chapter 7) will generate the structured YAML file (which describes the way how dialog will be reproduced) and sends to the server side (Chapter 7). After processing, server-side uploads file to the DGI server. At this moment this dialog is available for use with Alexa assistant. To achieve this goal it is needed to implement custom Alexa skill (Chapter 8) which is managed by AWS Lambda function (Chapter 9) where the dialog is processed and particular phrases send to Alexa. Alexa transforms text into voice format and here starts the communication between user and Alexa.

Chapter 6

Design

6.1 Design the Application

Design or design specification focuses on criteria to be satisfied in designing a product. Following the idea of application, the main frame should be divided into three parts:

- Left part – menu, where user can click on a particular item(type) and add to Central part.
- Central part – place, where we see edited current form. Selected items will be added at the end.

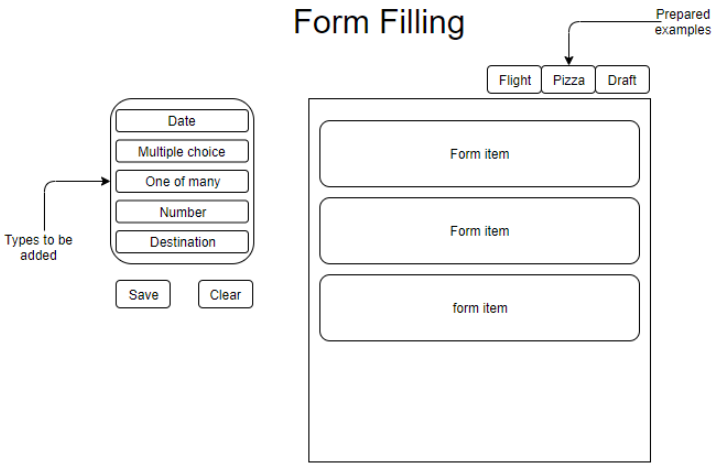


Figure 6.1: View of the main page. On the left, user selects inputs, these are added to the central part for editing.

6.1.1 Left part

In the left part, there are types of inputs, which can be clicked and added to central part.

Every item in the menu should contain Name to determine which item was selected.

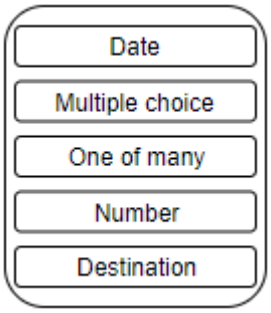


Figure 6.2: View of the type

Also, left part contains two main buttons, **Save** and **Clear**. By clicking the Save button, it will collect all of the inputs. When Clear button will be clicked, it should make the entire form empty.

6.1.2 Central Part

Central part represents the form, where input types from Left part would be added. In Central part, every form item should follow this structure:

- The head of form item should have:
  - Item type. Could be on of those :
    - Date
    - Destination
    - Checkbox
    - Radio
    - Number
  - Help information
  - Input question/phrase (changeable)
  - Menu with :
    - Button to add another item field.
    - Button to open configuration menu.
    - Button to delete an item.
- Body should contain item fields which are different for every item type.
- At the bottom of the particular item, there will be configuration menu with:

– Input for the user's question.
– Input for setting help information for a selected item.

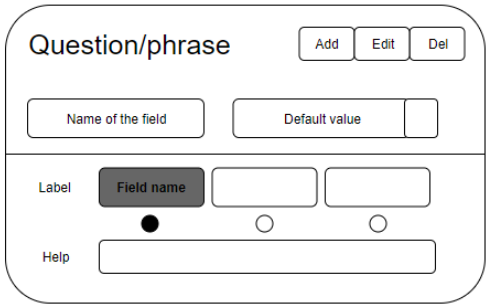


Figure 6.3: View of the form item

6.2 Dialog design in YAML

The DGI (Dialog graph interpreter) handles dialogs, represented in YAML. The DGI system is available online. The (YAML) dialog consists of states(nodes). The basic functionality is 'SAY' and 'GOTO'. 'GOTO' command has two variants:

1. transition to the next node,
2. first asking the user for response, then transit to the specified node. (See example in Chapter 8)

Dialog in YAML has three main nodes:

- WELCOMEPOINT
  This is where the run process will start. If user starts Alexa skill, the greetings from WELCOME POINT is used. After user response, the utterance is sent to ENTRY-POINT node.
- ENTRYPOINT
  If user starts with utterance (i.e "Alexa, ask skill for the weather" or Slack), the dialog starts by processing the user input.
- EXITPOINT
  Represents end of the conversation, says goodbye to the user and ends the dialog.

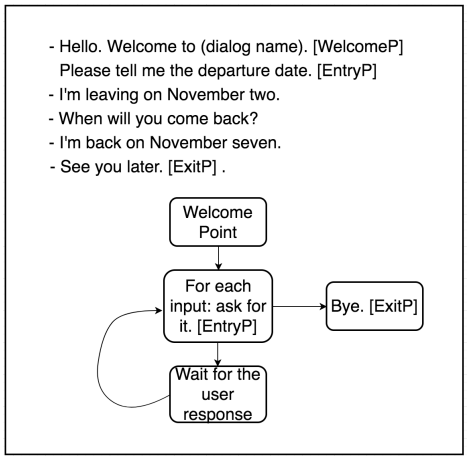


Figure 6.4: Example of YAML dialogue process used in our app, In square brackets we have shown the node types.

6.3 Process life cycle

The creation of voice form starts by creating the HTML-like form. Then it is exported and uploaded to online server (DGI). Afterwards, upon invocation of voice app (Alexa), DGI is queried and generates responses (questions) The whole process looks like this:

1. Steps from application point of view
   (a) Create a form using an application
   (b) Generate YAML file with steps depended on the User input
   (c) Upload YAML file to DGI
2. Speech processing steps
   (a) Alexa sends input to a Lambda AWS
   AWS Lambda is a compute service that lets you run code without provisioning or managing servers.
   (b) Lambda AWS communicates with DGI (user input, Alexa output)
   (c) DGI performs NLP (Nature-language processing), validation, entity recognition and saves data. Returns an answer to Alexa (via AWS Lambda)

For better understanding of individual steps and the life cycle of application, here is the approximate example.

1. User opens skill: "Alexa, open Form Builder"
2. Alexa finds the skill (by name)
3. Sends the "Launch intent" to AWS Lambda

4. Lambda contacts DGI (provides no user input)
5. DGI constructs welcome response (with the first question) and sends it back to Lambda
6. Lambda transforms it to the correct format (speechless) and sends it to Alexa
7. Alexa says welcome message (i.e.: "Welcome to FormBuilder. When are you leaving for the trip?")
8. User says: "I'm leaving tomorrow"
9. Alexa sends the utterance to Lambda
10. Lambda includes the user's utterance and sends it to DGI
11. DGI calls NLP: Intent detector.
12. NLP intent detector returns: intent = DEPARTURE_DATE
13. DGI calls NLP: Entity detector
14. NLP returns entity: type: DATE, value: 2018-11-22
15. DGI calls NLP: validation
16. NLP validation returns: OK or not OK
17. DGI sets the (form/dialog) variable: departure_date = 2018-11-22 if validation was OK
18. DGI constructs next question (or ends the dialog)
19. (a) DGI sends goodbye (if the form is complete) to Lambda (and signals the end of dialog.
    (b) DGI sends next question to Lambda. Then goes to step 6.

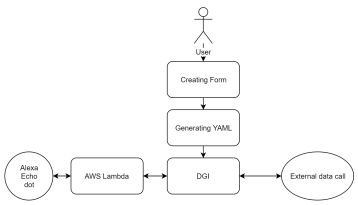


Figure 6.5: Process life cycle describing the flow from the user perspective: 1)form creation and upload; 2) dialog flow via Alexa.

6.4 UML class diagram

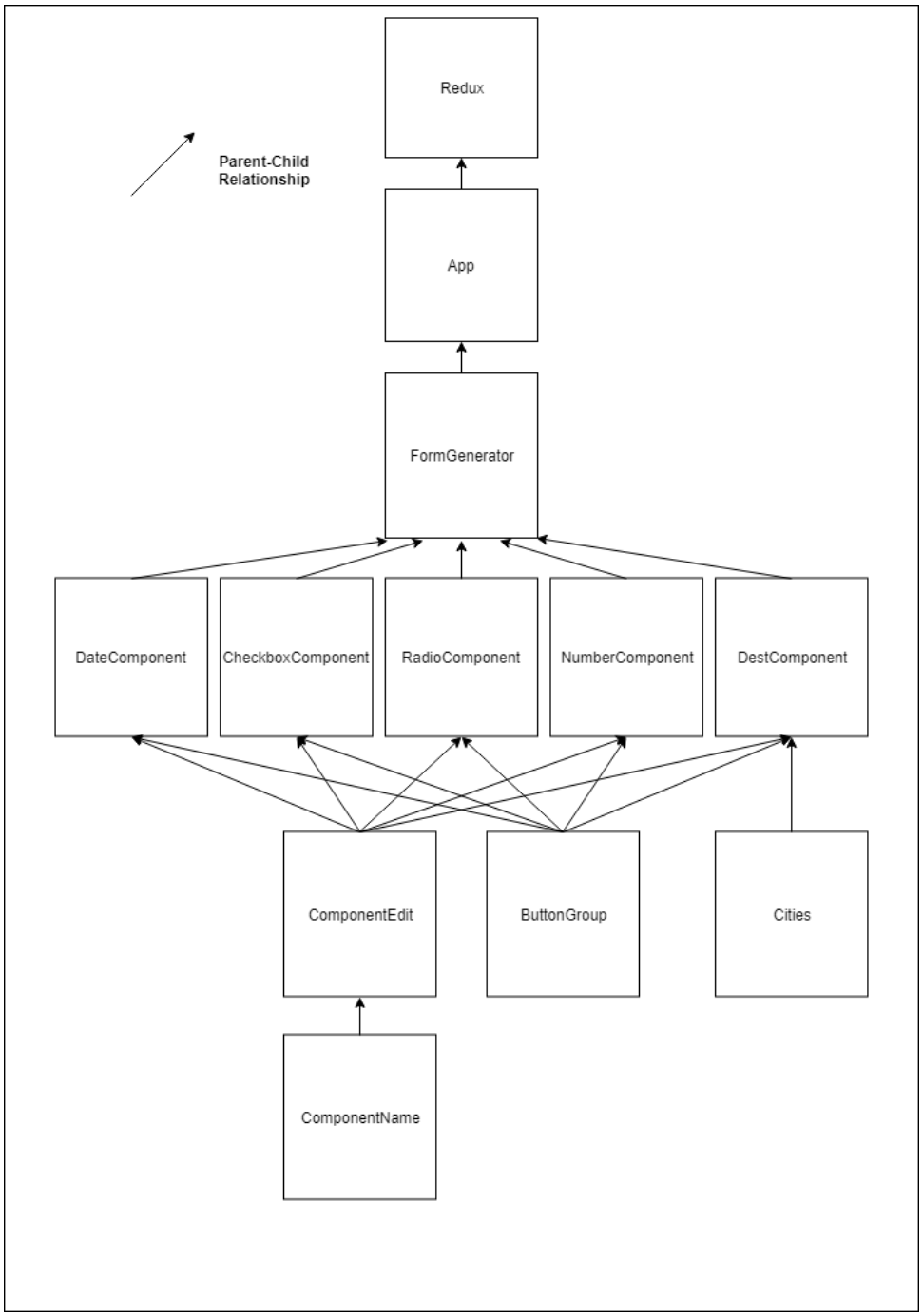


Figure 6.6: UML class diagram of the application. See description in chapter 6

Chapter 7

Implementation

The implementation part of this work will follow rules and contain all the requirements that were described in **Requirements** and **Design** chapter.

7.1 Structure of the application

ReactJs is working with components, component is a concept similar to class, when calling some particular component, it returns pre-designed HTML code.

React with Redux can be broadly broken down into:

- (store) Providers
- Container components
- Components

In this section, I will follow the UML class diagram (see Fig 5.6), where the structure was shown.

7.1.1 "Provider" Redux

Provider acts as a dependency container that holds the **store** and makes it available for the components to share the "state" of the application.

Store is a place where the current state of the application (with all related input, output variables) is saved. This means that every change that was done in some component (component A), saves to the store. Thanks to this, another component (component B), which doesn't have direct connection to the 'A', knows about the changes not from "Parent" component, but directly from store. (see Fig 6.1)

7.1.2 "Container" App.js

Container component is a component that is responsible for retrieving data, and in order to get that data, the component needs to use Redux's built in `connect` and `mapStateToProps` functions. App.js is a component, where all of the generated HTML-code from React will be saved.

7.1.3 "Component" FormGenerator.js

This component contains the main functionality of the application. It calls FormItem.js to create item form, saves created form and uploads generated YAML to DGI (requirement #4, #6 and #7), removes form (requirement #3).

7.1.4 "Component" DateComponent.js and other types

DateComponent.js represents the item that will be added to the form. Structure is the same as on Fig 5.3. Depending on input variables from parent component, it creates desired form item with the functionality described in requirements #2 and #6.

7.1.5 "Component" ComponentEdit.js

This component contains the editing menu which allows user manually set Question and Help for the form item.

7.1.6 "Component" ComponentName.js

ComponentName.js allows the setting of input Question.

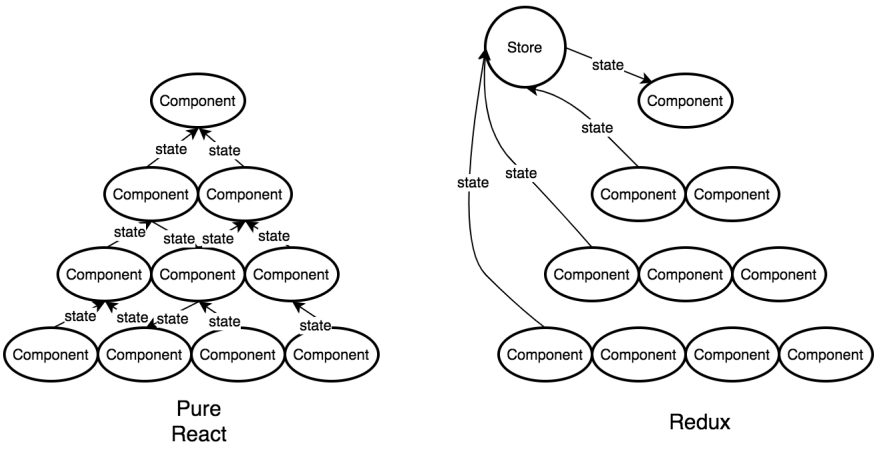


Figure 7.1: State sharing in React and React with Redux [14]. The main difference is that communication between components is going throw Store, so all of them have current state of the application.

7.2 Creating Form

Application allows user to create a custom form for representing user's requests.

The main file for this goal is **Form.js** which contains both **View** and **Controller logic**. This component represents the main frame in application. It contains **Menu** part with all of

the types, which could be added to form, **Form** part, where items will be saved and **Saved forms** part, where user can see already created forms.

An example is available in chapter 8.

7.2.1 Adding form item

Form items are declared as parts of the form list. When a user selects some of the types from the menu part, general count of items and count of selected item are incremented by one. This was done so that each element had a unique index. At the same time, it calls a **concat** method, which adds this particular item to the main list. During adding it calls sub-component to be created with three inputs:

- **key**: Unique key of the form list
- **id**: Count of already created elements of this type
- **index**: Unique number for deleting item correctly, **key** couldn't be used, because it can't be access from child component

```
case 'Date':
    this.setState({
        count: this.state.count + 1,
        numberOfDates: this.state.numberOfDates + 1,
        items: this.state.items.concat(<Date key={this.state.count} id={this.state.numberOfDates}
                                             index={this.state.count}
                                             onDelete={this.deleteItem.bind(this)}/>)
    });
    break;
```

Figure 7.2: Creating of form item

After, selected type calls a component to be created with pre-defined properties, which are common for all of the types, such as:

- count of input fields of this form item
- list of input fields
- tooltip value
- boolean value – collapse (which determines if configuration menu is open or not)
- boolean value – tooltipOpen (which determines if tooltip Value can be seen or not)

And structure, which was described in Design chapter.

7.2.2 Removing form item

In ReactJs it is not enough just to delete item from the list. It will only imitate deletion, but it still will be saved in list. In this case, developer has two choices:

1. The first one is to slice the list. By the **index** property you will find the index of item which is supposed to be deleted, slice the list before this index and after and connect this two parts.
2. The second one, which i prefer, is to filter the array. You create the instance of the existing list, call **filter** property of array and save all of the items, except item with specified index.
3. The third one is to mark form item as deleted (allows undelete action).

For this application we have used option number two.

7.2.3 Setting form item question

For a better user experience and more natural conversation with voice assistant I decided to implement an opportunity for setting custom question which user would be asked during filling form by his voice.

The structure contains three input fields with connected to them radio buttons. By selecting one of this buttons appropriate input will be shaded and value will be set to 'Field name' and after pushing 'Name' button, the full question is connected to this form item. Using this system, in generated YAML, every 'Field name' will be replaced with item field name of specific form item.

In below example can be seen the setting of question for a Date type of item.

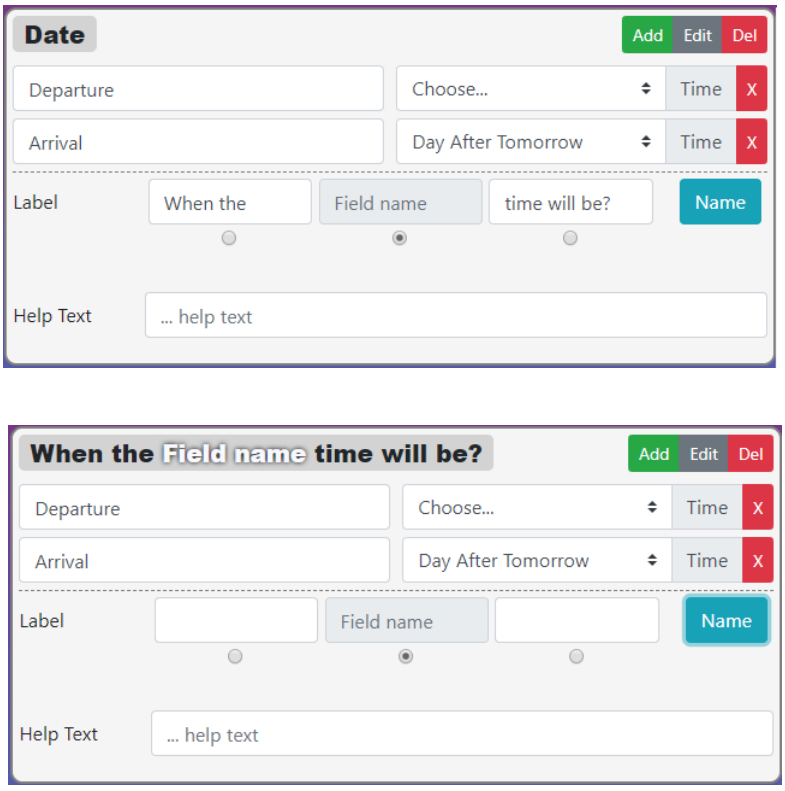


Figure 7.3: Structure of naming form item

7.2.4 Setting help to the form item

Help option supposed to add extra information about form item in text and voice format. In case of dialog, when a user doesn't know what to do, he can say 'help' and this option will be said.

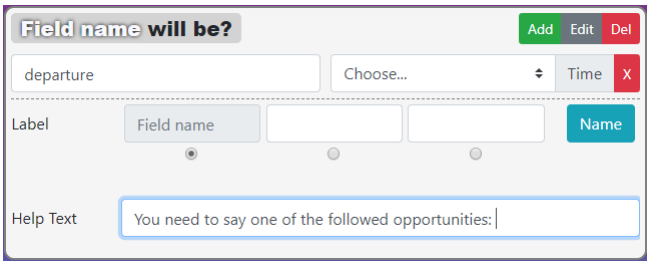


Figure 7.4: An example of setting help

7.2.5 Adding/removing form item input fields

In this work we are using standard HTML inputs [15]. For adding an input field, user should use the Add button in the upper-right corner of the form item. The process of adding and removing of input field is the same as adding and removing an form item. All inputs are added to the form item list of input fields with a unique key and number, which means the number of inputs in this list. Deletion occurs by index and filtering.

7.3 Deletion of form

Deletion sets all counts for form item types to zero. Also it sets list of form items to be empty. After this you can start from scratch in creating your custom form.

7.4 Export to YAML

After creating custom form and press submit button, export to YAML process starts. Firstly, in **Form.js** file it calls `createYaml()` function is called, which in turn calls sub-functions, which separately create the components of YAML file. YAML file is represented as `String` in the application.

It uses the following sub-functions:

- `createHeader()` contains header information, such as name of created form, date, author.
- `createWelcomeNode()` creates WELCOMEPOINT.
- `createInputsNode()` this creates node for every form input field, which user created. ENTRYPOINT.
- `createIntentEntityDetectorNode()` black box.

- `createExitNode()` creates EXITPOINT.

After all of the functions were processed, and YAML file was created, `fileDownload` function is called. It downloads created file with name consist of `nameOfCreatedForm` plus `.yaml` extension. It saves the YAML to the DB and upload to DGI via REST POST.

Chapter 8

Testing

Testing of this work was divided on two parts.

- Unit testing.
- Testing functionality of web application.
- Testing dialog with Alexa.

8.1 Unit tests

Unit testing involves breaking your program into pieces, and subjecting each piece to a series of tests.

For performing them I used Jest, JavaScript test runner. Those tests mounts a component and check that it doesn't throw an exception during rendering. Another type of tests controls Component's output. Basically, they controls the functionality of components functions.The last one is testing event. When a user submits the form, the listener will call following function only if the input is valid.

All unit tests are passing without errors.

8.2 Testing functionality of web application

We need to test functionality to make sure, that functional requirements have been done properly.

Manually I've tested creation of form and the following modifications (i.e set item id, add item sub-fields etc.) with form item, correctness of validation for form item fields, exporting to YAML and uploading YAML to DGI. Every functionality is working as expected.

8.3 Testing dialog with Alexa

To verify, that we achieved our goal and able to fill created form by voice, we created several trials of YAML file and upload them to DGI.

More about Alexa in chapter 8. Examples of correct and incorrect dialogs, also as an example of YAML file are available in appendix A.

8.4 Compliance with requirements

The following table shows the functional and non-functional requirements and their implementation state together with test result.

Table 8.1: Compliance with requirements

Functional requirements	Implemented	Tested
Add/remove input form item	✔	✔
Set input form item id	✔	✔
Set input form item the constraints	✔	✔
Set input form item help	✔	✔
Add input form item sub-fields	✔	✔
Remove form	✔	✔
Export to YAML	✔	✔
Set validation for form item fields	✔	✔
Process filled form (data)	✔	✔
Localization: only en_US	✔	✔
Browser Portability	✔	?[1]

[1] May not work with old version of IE

Chapter 9

Alexa

One of the main requirements of this work was an opportunity to fill the form by your own voice. To do this I have implemented custom Alexa skill powered AWS lambda function.

9.1 User - custom skill communication

The communication between the user and the custom skill is achieved via an Alexa powered device such as the Echo. Activation of a particular skill is done by saying the skill's invocation name along with the trigger word ("Alexa") which activates the Echo device. For example, the command "Alexa, open form filling" will start the "Form filling" skill.

Upon activating the skill, the user is able to send other voice commands, or speech requests, to the skill.

9.2 Skill structure

The Alexa skill consists of two main components: the skill interface and the skill service.

The skill interface processes the user's speech requests and then maps them to intents within the interaction model. The intents are actions that fulfill the spoken requests from the user. Every intent has at least one utterance which the user might say to invoke the intent. If a specific intent is detected, the skill interface creates a json encoded event, which is passed to the skill service.

The skill service determines what actions to take in response to the JSON encoded event received from the skill interface. Upon reaching a decision the skill service returns a JSON encoded response to the skill interface for further processing. After processing, the speech response is sent back to the user through the Echo.

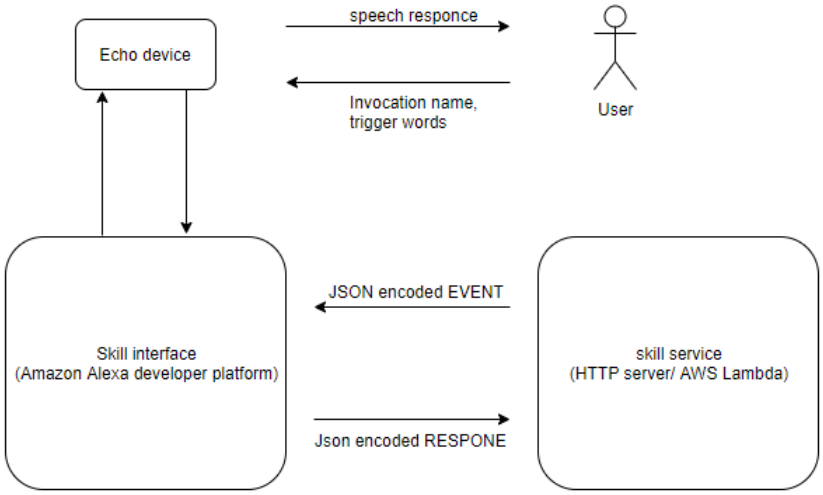


Figure 9.1: User-skill communication

9.3 Building custom skill

9.3.1 Building the Interaction Model

The Interaction model is a very important part of the skill. It contains the skill's invocation name, intents and utterances, etc., which are crucial for the interaction with the skill services.

Firstly it is needed to set up invocation name to start the interaction with our skill.

9.3.1.1 Intents and Slots

An intent represents an action that fulfills a user's spoken request. Intents can optionally have arguments called slots. A custom slot type defines a list of representative values for the slot. Custom slot types are used for lists of items that are not covered by Amazon's built-in set of types. When using a custom type, you define the type and its values, and specify the name of the type as part of the intent definition.

For our skill we needed:

- An intent that handles the user speech answer about Date. E.g. today, tomorrow, day after tomorrow, next week, next month
- An intent that handles the user speech answer about Yes/No question.
- An intent that catches when a user said 'help'.

```
"types": [
    {
        "name": "Date_Slot",
        "values": [
            {
                "name": {
                    "value": "next month"
                }
            },
            {
                "name": {
                    "value": "next week"
                }
            },
            {
                "name": {
                    "value": "day after tomorrow"
                }
            },
            {
                "name": {
                    "value": "tomorrow"
                }
            },
            {
                "name": {
                    "value": "today"
                }
            }
        ]
    },
    {
        "name": "Yes_No_Slot",
        "values": [
            {
                "name": {
                    "value": "Yes"
                }
            },
            {
                "name": {
                    "value": "No"
                }
            }
        ]
    },
```

(a) Intents

```
"interactionModel": {
    "languageModel": {
        "invocationName": "form filling",
        "intents": [
            {
                "name": "AMAZON.FallbackIntent",
                "samples": []
            },
            {
                "name": "AMAZON.CancelIntent",
                "samples": [
                    "goodbye",
                    "enough",
                    "stop",
                    "exit"
                ]
            },
            {
                "name": "AMAZON.HelpIntent",
                "samples": []
            },
            {
                "name": "AMAZON.StopIntent",
                "samples": [
                    "goodbye",
                    "enough",
                    "exit",
                    "stop"
                ]
            },
            {
                "name": "AMAZON.NavigateHomeIntent",
                "samples": []
            },
            {
                "name": "StartFormIntent",
                "slots": [],
                "samples": [
                    "Go form filling",
                    "Load form filling",
                    "Let's go",
                    "Start"
                ]
            },
```

(b) Slots

Figure 9.2: Interaction model example

9.3.2 Creating the AWS Lambda Function

The second part of user-skill communication is to create and configure the AWS lambda function. This function receives events from and returns responses to the skill interface. Events contain the information about the way in which the user is interacting with the skill including the type of request user has triggered.

There are three main request types:

1. LaunchRequest is sent within the event to the Lambda function when the user invokes the skill by saying its invocation name.

2. IntentRequest is sent within the event to the Lambda function when the user interacts with the skill, i.e. when his speech request is mapped to an intent. The intent is also inscribed in the event.

3. SessionEndedRequest is sent within the event to the Lambda function when the session ends, due to an error, when the user says “exit”, when the user doesn’t respond while the device is listening, or when the user says something that doesn’t match an intent defined in the skill interface while the device is listening.

Amazon provides several languages which you can use to program. I’ve selected Python because of the easy and clear code implementation and big base of great reviews from other users.

9.3.3 Alexa-AWS lambda connection

For the correctness work of implemented skill and clear communication between Alexa and AWS lambda, you need to share Alexa skill id to the AWS lambda function and AWS lambda function id to Alexa skill.

Chapter 10

Example dialog

10.1 Form creation

Here you can see the process of voice dialog creation on a trip planning task. The first step is to create a form.

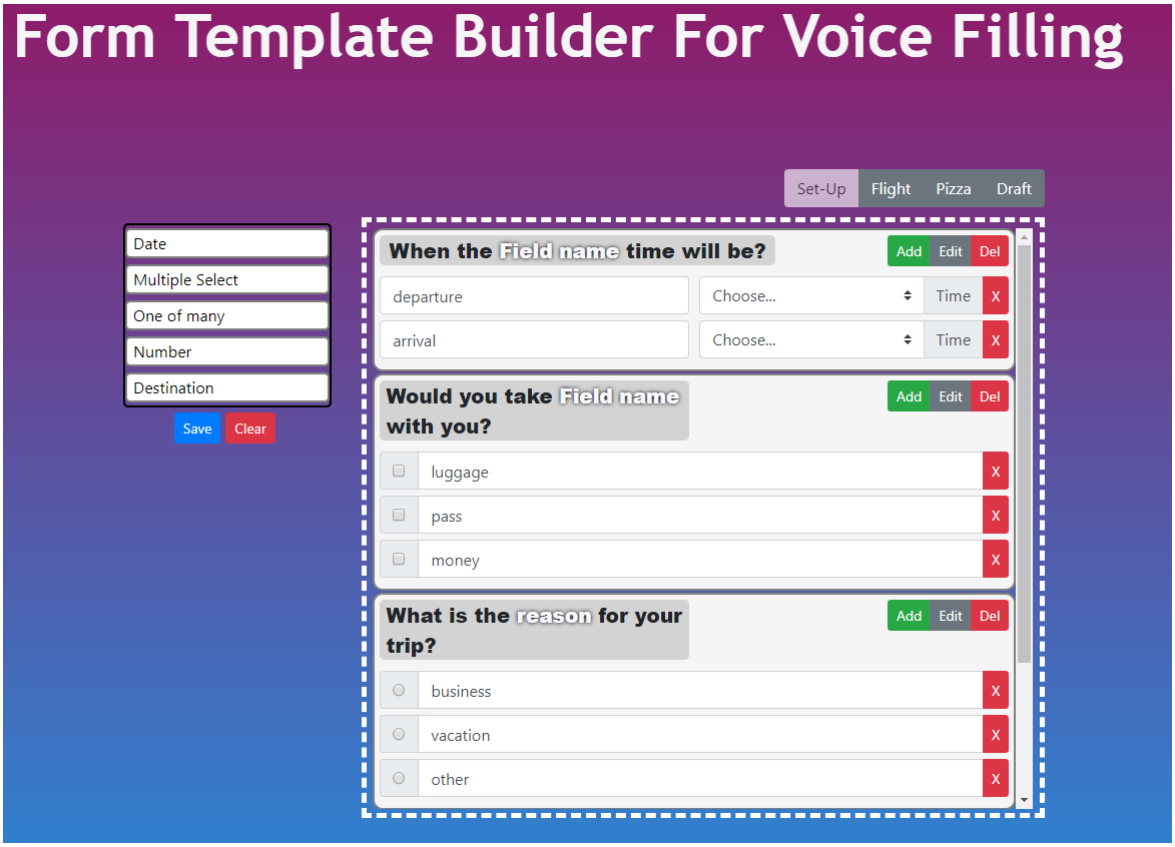


Figure 10.1: Example of form creation

We expect the user to input the departure and arrival date (in range of possible variants), multiple choice of items that he would like to take, one of many selection of the reason for this trip and two cities 'from' and 'to' user is going.

Second step is exporting to YAML

Listing 10.1: Pseudocode of generated YAML dialog

```
NodeList:

  // This node greets user
  welcomeNode:
    node_type: "WELCOMEPOINT"
    - action_01: "SAY"
      condition: "!avail('${global.f1.returning_user}')"
      reply: "Hello. Let's begin. "
      get_user_response: false
      goto_node: "mainFormNode"

 // Example of node where user will answer the question.
  departure:
    - action_01: "SAY"
      reply: "When the departure time will be? "
    - action_02: "GOTO"
      get_user_response: true
      goto_node: intentEntityDetectorNode

 // Summarize all of the information said by user and controls if it's right.
  endDialogNode:
    - action_01: "SAY"
      reply: "Your departure will be ${dialog.f1.departure}, and arrival will be
          ${dialog.f1.arrival}. Your selected items are luggage, pass, and money.
          You have preferred ${dialog.f1.reason} as reason. You have chosen
          ${dialog.f1.from} as from city, and ${dialog.f1.to} as to city. Is it
          alright?"
    - action_02: "GOTO"
      get_user_response: true
      goto_node: intentEntityDetectorNode

 // In case user wants to start filling the form from scratch, this node will
     clear all of the variables already defined.
  clearThisDialog:
    - action_01: "UNSET"
      unset_variable: "${dialog.f1.departure} IF avail('${dialog.f1.departure}')"
      ...
    - action_11: "GOTO"
      get_user_response: true
      goto_node: intentEntityDetectorNode

 // Managing node. Describes the flow of dialog.
  mainFormNode:
    node_type: "ENTRYPOINT"
    - action_01: "HELP_FOR_NEXT_NODE"
```

```
      help: "Say one of this options: today, tomorrow, day after tomorrow, next
          week, next month"
    - action_02: "GOTO"
      goto_node: "departure IF !avail('${dialog.f1.departure}') &&
          !avail('${dialog.f1.clearThisDialog}')"

// Node where all variables will be set to user answers.
 intentEntityDetectorNode:
    - action_01: "SET"
      set_variable: "${dialog.f1.departure} TO today IF dialog.userInputLast ==
          'today' && dialog.nodePrev == 'departure'"
      ...
    - action_25: "GOTO"
      goto_node: "mainFormNode"

// Indicates the end of dialog.
 exitNode:
    node_type: "EXITPOINT"
```

The last one is an example of dialog uploaded to DGI.

Listing 10.2: Dialog in command line. '-' generated text; '»' user input

```
- Hello. Let us begin.
- When the departure time will be?
    >> today
- When the arrival time will be?
    >> help
- Say one of this options: today, tomorrow, day after tomorrow, next week, next
    month
    >> tomorrow
- Would you take luggage with you?
    >> yes
- Would you take pass with you?
    >> yes
- Would you take money with you?
    >> help
- Say 'Yes' if you need this or 'No' in other case
    >> yes
- What is the reason for your trip? Business or vacation or other?
    >> vacation
- What is the from city you are going?
    >> Tokyo
- What is the to city you are going?
    >> Prague
- So, for the vacation trip today you will go to Prague from Tokyo and come back
    tomorrow. For this trip you will need pass and your luggage. Is it alright?
    >> yes
- Thank you. See you later.
```

Chapter 11

Conclusion and future Work

In this work, we focused on solving the problem of filling forms with a voice. We discussed a few different possible solutions that are available and focused on implementing a Web application to create dynamic forms and then use them in the environment of Alexa voice assistant.

11.1 Work accomplished

The main focus of our thesis was to implement a Web application for text into voice form transformation. We have done research of the already existed solutions to this problem, but due to non-compliance with the requirements, it was decided to implement our own application to fulfill the expectations.

After reviewing the existing chat solutions, we concluded that the most appropriate for us would be the Task-oriented dialog agents which is suited for short question-answer tasks.

The design of web application was provided in a simple way for user understanding and comfortable using.

For the implementation of the application we have chosen ReactJs library of JavaScript as client-side part which collects all of the information user gave to application and generates YAML dialog and Redux for managing the server-side tasks, such as sending Yaml dialog to the DGI server. More deep explanation is provided in Chapter 7.

To make sure that our application is working properly and generates a valid YAML dialog, we did several tests on an existing chat platform (DGI). Also we did tests with the Alexa voice assistant. The behavior was as expected and result could be seen in the demonstration example. The application is running locally. Source and instructions are available on gitlab.

11.2 Future extensions

Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods, or simply curiosity. There are some ideas that I would have liked to try when during the description and the development of the functionality in Chapter 6 and Chapter 7. This thesis has been mainly focused on the use implementing fully functional example of form

builder web application and completing more or less simple cases of tasks, leaving the study of advanced tasks outside the scope of the thesis. The following ideas could be done:

1. To make sure that this dialog participates only to you it is needed to implement user authentication with advanced user management.
2. It would be nice to share already created dialog with other users.
3. More flexible validation. Add constraints to particular fields.
4. Implement additional types that can be added to form.

Summarizing this, an extension to the application is the implementing more friendly interface and providing the way of completing more difficult task with Alexa voice assistant.

Bibliography

[1] Gupta Gaurav. *Mastering HTML5 Forms*. ebook, 2013.

[2] Alex Colon. *Amazon Echo*. PCmag.com, 2015.

[3] Dave Roos. *How Interactive Voice Response (IVR) Works*. ebook, 2016.

[4] Angular. "What is Angular?" In: (2018). URL: https://angular.io/docs.

[5] Ernando Chico. "What is Vue.js and What are its Advantages". In: (2018). URL: https://hackernoon.com/what-is-vue-js-and-what-are-its-advantages-4071b7c7993d.

[6] Clark Evans. "YAML Ain't Markup Language". In: (2018). URL: http://yaml.org/spec/1.2/spec.html.

[7] Gistia. "A Beginners Guide to Redux". In: *gistia.com* (2018). URL: http://www.gistia.com/beginners-guide-redux/.

[8] Daniel Jurafsky. *Speech and Language Processing*. Stanford University, 2018.

[9] Jérôme Macias. "Jest, The React.js Unit Testing Framework, In Practice". In: (2018). URL: https://marmelab.com/blog/2015/06/24/jest-in-practice.html.

[10] Jared Nance. "TypeScript vs. JavaScript". In: (2018). URL: https://stackify.com/typescript-vs-javascript-migrate/.

[11] Nitin Pandit. "What Is ReactJS and Why Should We Use It?" In: (2018). URL: https://www.c-sharpcorner.com/article/what-and-why-reactjs/.

[12] Robert R. Perkoski. *Introduction to Web Development*. ebook, 2018.

[13] Mikey Stecky-Efantis. "5 Easy Steps to Understanding JSON Web Tokens (JWT)". In: (2018). URL: https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec.

[14] Brad Westfall. "Learning react-redux". In: (2018). URL: https://css-tricks.com/learning-react-redux/.

[15] MDN web docs. "<input>: The Input (Form Input) element". In: (). URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input.