CoronaHackathon

wir

Mar 22, 2020

CONTENTS

As part of the WirvsVirus hackathon we join the fight against COVID-19.

The last couple of weeks were defined by the spread of the virus around the world. Europe, in particular, rapidly became the epicenter of the current epidemic. As a consequence, borders throughout Europe were closed in a rush leading to heavy traffic jams. Europe's free and borderless market, risks being halted due to merchants, commuters, and truckers being stuck attempting to cross a border.

For this weekend we chose to **hack** together a prototype app that allows officials to provide questionnaires digitally, enabling border crossers to answer questions in advance and reduce the work of border policemen. Checkout our video on YouTube to see what we have achieved!

CHAPTER

ONE

INTRODUCTION

Consider the following user story:

A truck driver delivering goods throughout Europe

On his journey, the truck driver travels from country A to B but does not speak the language spoken in country B fluently.

How can the time of the border crossing be minimized?

A policeman of country B at the border will want to know the truck driver's identity, his destination, and reasons for travel. Most of the time will be lost communicating and answering questions while overcoming the language barrier.

Note: What if the truck driver could answer the questions in advance, **in his own language**?

The time it would take for the policeman to verify the answers would be only a fraction of the time it currently takes and seriously reduces traffic jams all over Europe's closed borders.

Fig. 1: Mock Up of the questions a person crossing the border from Germany to Switzerland might answer answer using our application.

CHAPTER

TWO

APPLICATION

We took this scenario as a guideline for our application and decided to write the following tools:

- A questionnaire generator allowing officials to upload and distribute questionnaires to people who want to cross their border.
- A client webapp that allows anyone to answer the questions in their own language and generate a QR-Code containing the answers as a URL.
- A website containing the answers in the policeman's language.

Therefore allowing policemen to get answers to all questions they are supposed to ask just by scanning a QR-Code!

In the following the details of our implementation are documented.

2.1 Questionnaire Generator

The questionnaire generator is a simple qt based python program that allows officials to generate questionnaires in the shape of a JSON File. The JSON File contains questions and language information.

2.1.1 project.dict_generator module

Summary

Functions:

add_lang_dialog	Add language dialog to left window.
cancel_inspect_mode	
change_lang_inspect_mode	Go into inspection mode.
change_option	Change view.
close_application	
create_new_dict	Initialize a new dict.
create_table	Create a new table.
fill_new_language	Fill new language adding it to the dict_state.
finish_new_language	Forward button when adding new language.
handle_table_click	
new_dict_begin_dialog	Open dialog to add a new dictionary and start with entering a new language.
new_dict_dialog	Switch to new dict dialog window.
next_new_language	Forward button when adding new language.

Continued on next page

Table 1 – continued from previous page

open_dialog	Open a file dialog to read json dict from file.
prev_new_language	Back button when adding new language.
save_dialog	Open a file dialog to save json dict to file.
set_left_window	Go into inspection mode.
update_lang_inspect_table	Update language table.
updated_table	Updated table.
write_to_dict	Write

Reference

project.dict_generator.**save_dialog**()
Open a file dialog to save json dict to file.

project.dict_generator.**open_dialog**()
Open a file dialog to read json dict from file.

project.dict_generator.**add_lang_dialog**()
Add language dialog to left window.

project.dict_generator.**new_dict_dialog**()
Switch to new dict dialog window.

project.dict_generator.**new_dict_begin_dialog**()
Open dialog to add a new dictionary and start with entering a new language.

project.dict_generator.**create_table**()
Create a new table.

project.dict_generator.**updated_table**()
Updated table.

project.dict_generator.**fill_new_language**()
Fill new language adding it to the dict_state.

project.dict_generator.**prev_new_language**()
Back button when adding new language.

project.dict_generator.**next_new_language**()
Forward button when adding new language.

project.dict_generator.**finish_new_language**()
Forward button when adding new language.

project.dict_generator.**handle_table_click**(*row*, *column*)

project.dict_generator.**set_left_window**(*idx*)
Go into inspection mode.

project.dict_generator.**create_new_dict**()
Initialize a new dict.

project.dict_generator.**change_option**()
Change view.

project.dict_generator.**change_lang_inspect_mode**(*row*, *colum*)
Go into inspection mode.

project.dict_generator.**update_lang_inspect_table**()
Update language table.

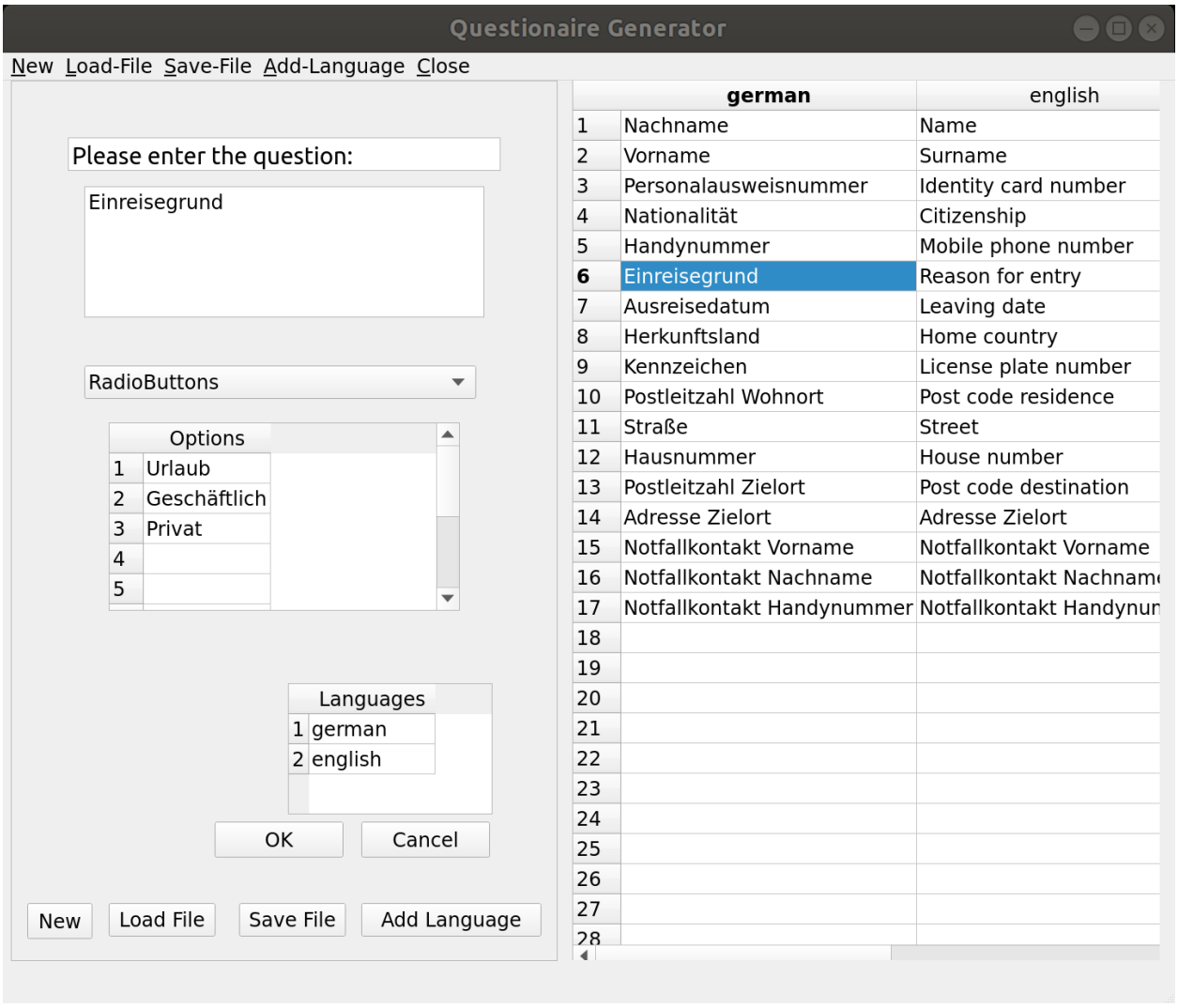

Fig. 1: Image of the Questionnaire Generator

project.dict_generator.**cancel_inspect_mode**()

project.dict_generator.**write_to_dict**()
Write

project.dict_generator.**close_application**()

2.2 Border Crosser Webapp

The border crosser's web app consists of only one website that, once opened, downloads the correct questionnaire and allows us to generate the resulting QR-Code without any further internet connection.

Our current state looks like this:

Fig. 2: Image of the Border Crosser App

This is how we imagine the app to be used on a mobile device:

2.3 Border Police Website

Encoded in the QR code is an URL containing all answers of the border crosser. The border police can then simply scan the QR code and open the website to see the answers on her/his device.

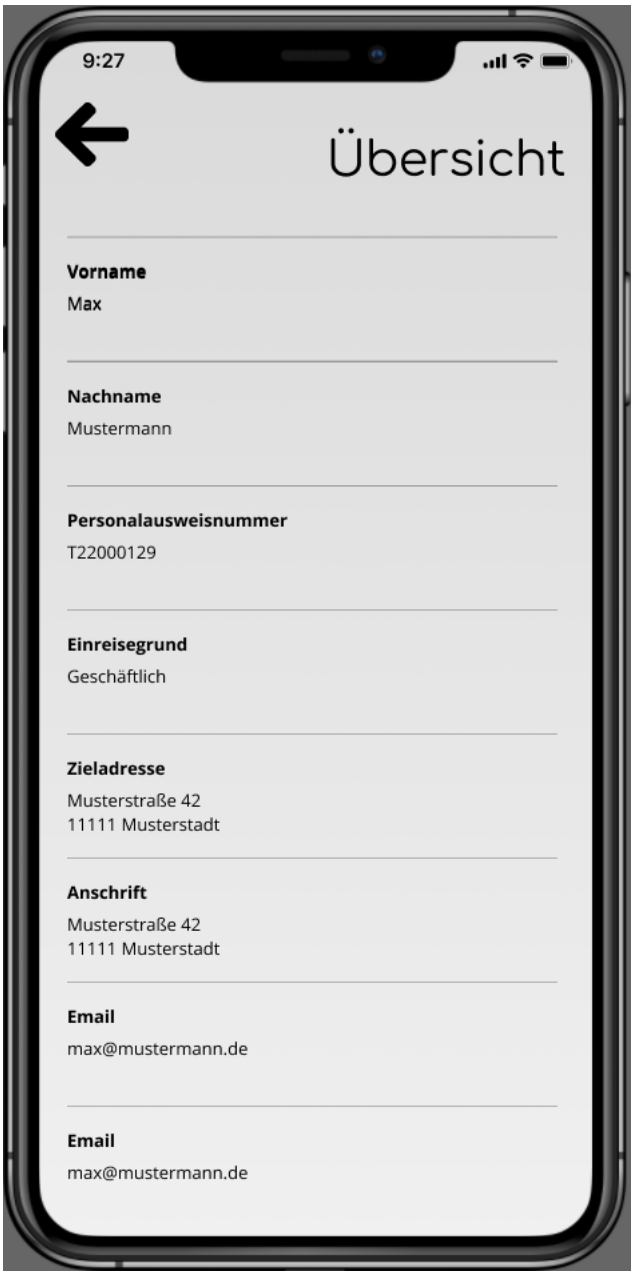

Fig. 3: Image of the Police Website

2.4 Backend: Connecting the Pieces

A flask webserver connects the individual parts of our application. A flask web server is used to serve the questionnaire to border crossers and answers the requests that are triggered when the border police scan's a QR code.

2.4.1 project.questionaire module

Summary

Classes:

LocalizedQuestion	hols a localized question
Question	hold the description of a question
Questionaire	holds all relevant data of a questionaire

Reference

class project.questionaire.**Questionaire**(*global_id*, *language_map*, *questions*)
Bases: object

holds all relevant data of a questionaire

global_questionaire_id = ''

__init__(*global_id*, *language_map*, *questions*)
Initialize self. See help(type(self)) for accurate signature.

language_map = {}

questions = []
initialization

:paran str global_id – the globally unique identifier of this questionaire. This is needed to connect questionaires with anwsers. :param dict language_map – (dict of str: (str: str)) – containes the text-snippets used in this questionaire in every supported language. The mapping is "language" -> "id" -> "text :param list questions – (list of Question)the actual questions in the questionaire

localized_questions(*language*)

class project.questionaire.**Question**(*question_id*, *anwser_type*, *options=[]*)
Bases: object

hold the description of a question

__init__(*question_id*, *anwser_type*, *options=[]*)
Initialize self. See help(type(self)) for accurate signature.

question_id = ''

anwser_type = ''

options = []
initialization

:param str question_id – the identifier of the question. This is needed to idenitify the corresponing question test stored in the Questionaire'S language_map and associate questions with anwsers :param str anwser_type – which kind of anwser is expected (string, date, PLZ,...) :param list options – optional (array of strings) specifies all possible anwsers

class project.questionaire.**LocalizedQuestion**(*question*, *question_text*, *options_texts*)
Bases: *project.questionaire.Question*

hols a localized question

__init__(*question*, *question_text*, *options_texts*)
Initialize self. See help(type(self)) for accurate signature.

question_text = ''

options_texts = []
initialization

:param Question question – the question :param str question_text – the text of the question :param options_texts – (list of str) the texts of the options

2.4.2 project.questions_from_json module

Summary

Functions:

read	Reads questionaire from file

Reference

project.questions_from_json.**read**(*filename*, *context*)
Reads questionaire from file

:param filename – path to file containing questionaire :rtype questionaire.Questionaire :return a questionaire

2.4.3 project.flaskr package

Submodules:

project.flaskr.app module

project.flaskr.config module

Summary

Reference

Subpackages:

project.flaskr.routes package

Submodules:

project.flaskr.routes.overview module

Summary

Functions:

get_overview_blueprint

Reference

project.flaskr.routes.overview.**get_overview_blueprint**(*questions*)

project.flaskr.routes.questionaire module

Summary

Functions:

get_questionaire_blueprint

Reference

project.flaskr.routes.questionaire.**get_questionaire_blueprint**(*questions*)

project.flaskr.services package

Submodules:

project.flaskr.services.forms module

Summary

Reference

PYTHON MODULE INDEX

p

INDEX

Symbols

A

C

F

G

H

L

N

O

P

Q

R

S

U

W